The Digital Dissector: A Study of Design Issues in Educational Multimedia

by

Christopher A. Horton

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering and Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 16, 1994

© Copyright Christopher A. Horton, 1994. All Rights Reserved.

The author hereby grants to M.I.T. permission to reproduce and to distribute copies of this thesis document in whole or in part, and to grant others the right to do so.

Author .................................................................

Department of Electrical Engineering and Computer Science

May 16, 1994

Certified by .........................................................

Wilson C. Hayes, Ph.D.
Mueller Professor of Biomechanics
Department of Orthopaedic Surgery, Harvard Medical School
Division of Health Sciences and Technology, MIT
Thesis Supervisor

Accepted by ............................................................

F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses
Department of Electrical Engineering and Computer Science
Abstract

This thesis examines the design and implementation of two educational multimedia modules that are intended to help medical students learn anatomy and dissection. The primary goal of both modules is to facilitate the completion of a dissection laboratory by presenting students with dissection instructions and elaborative material such as pictures and animations. The modules are also intended to be extremely user friendly and easily expandable. The structure of the modules is linear and rigid compared to most other multimedia programs because they are designed to minimize cognitive overhead and restrict users to single page digressions. The software also provides information regarding common surgical techniques, a quizzing facility, an annotation feature, and a usage tracking feature.

Wilson C. Hayes, Ph.D.
Mueller Professor of Biomechanics
Department of Orthopaedic Surgery, Harvard Medical School
Division of Health Sciences and Technology, MIT
Thesis Supervisor
This thesis is dedicated to my parents, Phil and Katy Horton who graciously funded my college education.

Acknowledgments

Thanks to Dr. Peter Ratiu and the Digital Anatomist Group at the University of Washington who provided me with ten of the animations in my software.

Thanks to Michael Harrington who provided advice and support.

Thanks to the Orthopaedic Biomechanics Lab of Beth Israel Hospital for letting me use your equipment.
Table of Contents

1 Multimedia and Educational Multimedia ................................................. 6
   1.1 Introduction ................................................................................. 6
   1.2 Definitions .................................................................................. 7
   1.3 The Structure of Multimedia Documents ...................................... 9
   1.4 Educational Multimedia .............................................................. 11
   1.5 Some Educational Multimedia Applications .................................. 12
2 The Digital Dissector Design, Part I .................................................. 17
   2.1 Background .................................................................................. 17
   2.2 Primary Design Goals ................................................................. 19
   2.3 Information Issues ....................................................................... 20
   2.4 Module Structure ......................................................................... 21
   2.5 Preliminary Objections ............................................................... 27
3 The Development Environment ............................................................ 32
   3.1 Hardware .................................................................................... 32
   3.2 Software: A.D.A.M ..................................................................... 32
   3.3 Supercard ..................................................................................... 33
4 Interface Design .................................................................................. 38
   4.1 Page Layout ................................................................................. 38
   4.2 Interface design .......................................................................... 38
   4.3 The Menu and Help System ...................................................... 41
5 The Larynx Module ............................................................................ 43
   5.1 Module One, The Larynx ............................................................ 43
   5.2 Additional features ..................................................................... 52
   5.3 Problems with the Larynx module .............................................. 57
6 The Knee Module ................................................................................ 64
   6.1 Introduction ................................................................................ 64
   6.2 Design ......................................................................................... 64
   6.3 Knee Module Features ............................................................... 66
   6.4 Code Improvements ................................................................... 72
7 The Digital Dissector: Discussion and Conclusions .............................. 76
   7.1 Introduction ................................................................................ 76
   7.2 Computer Related Design Issues .............................................. 76
   7.3 Hypermedia Issues .................................................................... 78
   7.4 Educational Issues ..................................................................... 82
   7.5 Advice for Other Developers ................................................... 83
   7.6 Future Extensions ...................................................................... 85
   7.7 Conclusion ................................................................................ 86
Appendix A ........................................................................................... 88
Appendix B ............................................................................................ 106
Appendix C ............................................................................................ 119
Appendix D ............................................................................................ 127
Bibliography .......................................................................................... 130
List of Figures

Figure 2.1: An Unstructured Module ....................................................... 22
Figure 2.6: A Highly Structured, Linear Module ........................................ 25
Figure 2.9: The Digital Dissector Structure ............................................. 26
Figure A.1 The A.D.A.M. Interface ....................................................... 89
Figure A.2 The Larynx Module Main Menu ........................................... 90
Figure A.3 The Larynx Module Introduction Page .................................. 91
Figure A.4 A Larynx Module Dissector Page ......................................... 92
Figure A.5 A Larynx Module Atlas Page ............................................... 93
Figure A.6 The Larynx Module Help Page One ...................................... 94
Figure A.7 A Knee Module Introduction Page With Annotation Field On .... 95
Figure A.8 A Knee Module Dissector Page .......................................... 96
Figure A.9 A Knee Module Common Procedures Page ......................... 97
Figure A.10 A Knee Module Atlas Page .............................................. 98
Figure A.11 The Knee Module Quiz Page ........................................... 99
Figure A.12 The Knee Module Programmers Page .................................. 100
Figure A.13 The PICT Viewer .......................................................... 101
Figure A.14 An Animation, Courtesy of Dr. Ratiu ................................. 102
Figure A.15 An Animated Scan, Courtesy of Dr. Ratiu ......................... 103
Figure A.16 The Director Movie Viewer ............................................. 104
Figure A.17 The Template Module Menu ............................................ 105
Chapter 1

Multimedia and Educational Multimedia

1.1 Introduction

Until recently, desktop computers were too slow to display digitized movies at acceptable speeds. The specialized hardware necessary to display animation or video often cost as much as the computer itself, making widespread use of high-end graphics only a dream. However, thanks to the furious pace of technology and the competitive pricing of current systems, nearly every computer sold today comes with graphics and sound capabilities that would put the high end system of yesterday to shame. The availability of this technology coupled with the growing popularity of CD ROMS has caused an explosion of multimedia computers and software.

While multimedia is popular as an entertainment medium, it also holds great promise as an educational tool. Computer based training has been found to improve retention and steepen learning curves. In addition, multimedia allows students to efficiently browse through large databases and access dynamic forms of information that would be impossible to put in conventional textbooks. This makes it useful in the classroom and powerful as a general educational tool.

This thesis centers on a pair of educational multimedia modules that I wrote to help medical students learn anatomy and dissection. In the course of discussing the development of these modules, I will provide the reader with a survey of important issues relating to the design of highly structured educational multimedia programs like my own.

---

1. see Shute & Regian, and studies cited in Nielsen,
This thesis can also be used as a reference for creating the remaining modules of the “Digital Dissector.” Chapter 1 introduces the reader to the basics of hypertext, multimedia and educational multimedia. Chapter 2 presents background information on the software and addresses my high level design goals and decisions. Chapter 3 provides an overview of the development environment used to create the software. Chapter 4 finishes up the discussion of module design. Chapters 5 and 6 describe the implementation of the two modules. Chapter 7 discusses a few other design issues and provides some suggestions for future work. A collection of pictures and a partial code listing are included in the appendix.

1.2 Definitions

1.2.1 Hypertext

hyper- pf prefix [ME iper-, fr. L hyper-, fr. Gk, fr. hyper - more at OVE]
1: above: beyond: SUPER- {hyperphysical} 2a: excessively {hypersensitive}
2b: excessive {hyperemia} 3: that is or exists in a space of more than three dimensions {hypercube} {hyperspace} - Webster’s Dictionary

The earliest ancestor of the modern multimedia application is what is known today as hypertext.¹ Hypertext is simply a computerized document that allows the reader to call up other related texts. For instance, a hypertext document may have a number of footnoted references like a regular document but would allow the user to browse through the text of those references without any more effort than a mouse click or a few key strokes. From a functional standpoint, a good analogy for the typical hypertext session might be a library search. You begin by reading one book, but notice an interesting footnote on one of the pages. You look up the book referenced in the footnote and skim through it. One of the chapters mentions an interesting idea, so you look up the books cited in the chapter

¹. Woodhead, p 6
bibliography. Your search for information continues through the library in this manner, and information from one source refers you to other sources. In a similar fashion, hypertext provides the user with a network of linked documents and a method for going from one document to another. The difference is that browsing and searching though a hypertext document is orders of magnitude faster and more efficient than browsing through a library.

1.2.2 Hypermedia and Multimedia

Once computers were able to display graphics, the first hypermedia documents were developed\(^1\). The only real difference between hypermedia and hypertext documents is that hypermedia documents are not constrained to pure text. Instead they may contain many forms of information including text, graphics, pictures, sound, animation, and video clips. The difference between hypermedia and multimedia is much more subtle, but only exists in a non-computer sense.

The term multimedia describes anything (project, performance, or software) that uses “multiple media.” This would include sales presentations that make use of films, computerized “guided tour” programs (where the user has little or no control), or multimedia software. Currently, the majority of multimedia applications are presentation tools, games, databases, and educational programs. Major software labels have been producing multimedia presentation tools and games for some time, but the educational market is just starting to take off. Use of multimedia as a global information retrieval and communication tool is also becoming popular as demonstrated by the rapid growth of the World Wide Web and Mosaic software.\(^2\)

Software manufacturers have been selling both multimedia and hypermedia programs as “multimedia,” effectively negating any differences between the two words, at least in a

---

1. Woodhead, p 7
computing context. Today you will only see advertisements for “multimedia” software. “Hypermedia” is almost never mentioned except in academic and technical literature. Throughout this thesis, I will use the two terms interchangeably.

1.3 The Structure of Multimedia Documents

Both hypertext and multimedia documents have the same high level composition, each consisting of chunks of information and paths between those chunks. The technical terms used to describe the parts of a hyper-document are based on the idea of an information network. The chunks of information are called nodes, and the paths between nodes are called links. A network of nodes and links is known as a collection, network, or module. Since “collection” does not stress an underlying unifying principle and “network” has structural implications, I will use the term “module” throughout this paper.

1.3.1 Nodes

The most important part of a hyper-document is the node which consists of bits of text, graphics, video, sound, or other information. A node might be a fragment of text (e.g., a quotation), a picture, an animated sequence with some explanatory text, an entire book, or anything in between. A single node can be thought of as a “page” in the hyper-document, and nodes are usually the smallest piece of information presented to the user. In order to emphasize the idea that each node is a separate packet of information, two nodes are almost never displayed in the same window at the same time. In systems that use multiple windows this still holds true, although more than one node (window) may be on screen at the same time. Using a house analogy, a node can be thought of as one room and its contents. The room may contain everything from a lone scrap of paper to a full set of furniture.

---

1. Jonassen & Grabinger, p 5
The size or *granularity* of a node usually depends both on the type of information being presented and the goal of the hyper-document. Most of the time the system designer would like to present a specific group of concepts to the user, so node granularity reflects the conceptual level the designer had in mind. In some cases, it may be desirable to have nodes of varying granularity so that the user can delve into greater detail or explore higher level concepts. Using the house analogy, granularity can be thought of as the size of the rooms.

1.3.2 Links

A *link* establishes the relationship between two nodes and is used to transport the user from one node to another.¹ In terms of the house analogy, a link is a teleporter or passageway from one room to another. Note that besides being used to go between adjacent rooms, teleporters can be used to travel to a room on the other side of the house or even to a room in a different house. Similarly, links can take the user to any page in the current module or even to a page in a different module.

In hypermedia software, links are typically embedded into the text and graphics of a node, but can also be independent of the node’s informational content (e.g. in the form of a button or menu). There will usually be some sign of where embedded links are, and a common technique for indicating textual links is to bold face or underline the linked word or phrase. Most links take the user to one specific destination node, but this does not always have to be the case; it is possible for a link to be context sensitive and jump to different nodes depending on the user's previous actions. Since today just about every computer has a pointing device such as a mouse, the most common way to activate a link is by clicking on it.

¹ Jonassen & Grabinger, p 6
Browsing is the term commonly used to describe the standard method of using a hyper-document, where the user starts at some page and proceeds to travel through the nodes of the document via links, following some interesting line of thought. Depending on how nodes are interconnected, a hyper-document may have many paths through it. Because of this path variability, it is common for modules to have a large number of links so that the user has a better chance of visiting every important node while browsing.

1.3.3 Modules
A module is a collection of related nodes linked together. A module can be considered a complete hyper-document by itself but may also be linked with other modules to form a larger document. For instance, a group of nodes might be collected and connected together to construct an American History module which could then be used to form a World History module. The important thing is that there is a high level unifying subject among the nodes of a module, in this case, American history.

1.4 Educational Multimedia
While presentation tools and games are certainly useful, multimedia can make the biggest impact in the area of education. The educational variety of multimedia is the same as any other kind except that its focus is on transferring useful knowledge to the user. As teachers and professors know, keeping the interest of students is as much of a challenge as getting them to learn and remember material. But it is exactly these two points that multimedia can be used to address. Among the other advantages of educational multimedia software are

1. Lessons become more interesting to the student. If nothing else, graphics, animation, sound, etc., make the software more likely to hold the user’s attention.
2. The students can learn at their own pace and target individual needs. In conventional instruction, all students are taught the same material and are required to do the same work. By taking advantage of the multiple paths through a hyper-document, a student can bypass areas that are already familiar. By the same token, students can delve deeper into areas in which they are weak.
3. Information is easier to absorb and retain when it is presented in more than one format. Studies have also shown that student performance is directly related to the amount of interactivity with the system.¹

4. Lessons become richer because more modes of expression are available. A prime example of this is reading Shakespeare. To the first time reader, Shakespeare may seem dry and boring, but when the text is augmented with spoken dialogue and videos of plays being performed, the works suddenly take on a whole new life.

1.5 Some Educational Multimedia Applications

Research in the area of educational multimedia is growing rapidly. However, the field is still in its infancy and tools and methodology are still emerging. As of yet, only a small number of development tools exist, and none are very good. Therefore, as work in the field progresses, the quality of applications will undoubtedly improve. Because addressing the state of the art would be a thesis by itself, I will only present enough information to give the reader an impression of current educational and medical multimedia. The Internet sources cited in this section provide an excellent starting place for a more in-depth inquiry.

1.5.1 The WWW

Arguably the most exciting hypermedia development is the World Wide Web, or WWW as it is more commonly known. The WWW is a global network of hypermedia documents linked together and based on a simple scripting language that allows users to quickly write up their own nodes or “pages.” This network is accessible and expandable by anyone with an Internet connection. The information on this network includes on-line magazines, weather maps, virtual libraries, art galleries, and commercial advertising. On June 1, 1993 there were approximately 130 WWW sites on the Internet and in less than a year that number has increased to about 1300² (note that one site may store a large number of pages). Because of the amount of information contained in the network, the WWW is also a powerful tool for learning. A comprehensive list of general multimedia sites is

¹ Silva, p 145
² http://web.mit.edu/afs/sipb/user/mkgray/ht/web-growth.html
available at http://cui_www.unige.ch/OSG/MultimediaInfo/index.html. Here is a small
sampler of WWW pages that are related to education:

   ANU Educational Material
2) http://utis179.cs.utwente.nl:8001/esperanto/hypercourse
   Esperanto Hypercourse
3) http://www.uji.es
   Department of Education - Universitat Jaume I, Castello, Spain
4) http://english-server.hss.cmu.edu/langs.html
   Languages and Linguistics
5) http://eryx.syr.edu/COWSHome.html
   ASKEric
6) http://www.exploratorium.edu
   The Exploratorium - Museum of Science, Art, and Human Perception
   with over 650 “hands-on” exhibits.
   Australian National University Science and Math resources for Education
   Science Education Dept. at Harvard-Smithsonian Center for Astrophysics
9) http://ultralab.anglia.ac.uk
   ULTRALAB - Anglia Polytechnic University - Learning Technology Research Center.
10) http://www.ed.gov
    U.S. Department of Education
    Educational Technology Branch of the National Library of Medicine
    (this site contains a database of multimedia medical programs)
12) http://hub.terc.edu
    TERC Science and Math Education
13) http://tecfa.unige.ch/info-edu-comp.html
    WWW Virtual Library of Educational Technology

1.5.2 Medical Multimedia

Because many groups involved in medicine already have high power computers for
imaging work, the medical field is one of the fastest growing educational multimedia
software markets. This is not to say that the software currently available is especially high
quality or widely used. Instead, a number of sites are developing customized applications
to suit their own needs and distributing this software to other groups. In addition, some
people have developed animated videos and visual databases which can usually be
purchased by other research groups. As multimedia techniques and tools develop, the medical industry will probably be among the first to benefit.

A survey of groups working on medical multimedia would take many pages, so instead I will briefly mention two of note. Each of these groups has produced a number of programs, some of which are available to outsiders. A relatively long listing of commercially available educational medical software is accessible on the Internet via Mosaic at http://www.etb.nlm.nih.gov/coursedb/methods.index.html. Many other interesting medical sites can be found on the net, including the Virtual Hospital (http://vh.radiology.uiowa.edu/) which has a number of on-line multimedia “textbooks.”

1.5.3 SUMMIT

The Stanford University Medical Media and Information Technologies (SUMMIT) group exists to help incorporate new technology into the classroom. Under the directorship of Dr. Parvati Dev, SUMMIT’s primary role is to harness the power of computers as educational tools and resources. Along these lines, a number of multimedia projects aimed at medical students are currently under development, including “The Anatomy Lesson” which is designed to teach basic anatomy to first year medical students. Here is a description of the program:

“The program uses the format of a discussion section with questions, diagrams, interactive images, and animations to draw students in and maintain their interest. The interactive labeled images and diagrams allow students to learn and to test their knowledge by actively seeking out information. Leading questions focus the student on key structures and systems. An enormous database of multiple choice questions allows students to test their knowledge as they learn. Animations are used to illustrate difficult concepts.”

In a personal communication, Phil Constantinou, one of the programmers involved with The Anatomy Lesson, said that a number of studies have been performed with varied success rates. He did not reference any specific papers, but mentioned that in one study half of an anatomy class used the software and the other half did not. It turned out that test scores were not significantly different between the two groups. He said that other projects had shown better results but did not mention them by name.

Two completed SUMMIT projects include BrainStorm and Walk About. BrainStorm is a highly interactive program designed for use in the teaching and review of human neuroanatomy. It features four different presentations of neuroanatomical structures including cross sections, dissections, diagrams, and textual explanations. These four modes are interconnected so that students can efficiently examine all aspects of a particular region. Walk About is primarily a quiz program, being designed to simulate the Stanford medical exams. The program features interactive images, multiple choice and free response questions, and offers a mode for learning and a mode for testing. This software is popular with the Stanford medical students.

1.5.4 The Digital Anatomist

The Digital Anatomist group in the Department of Biological Structure at the University of Washington is developing a multimedia system intended to teach gross anatomy. Two products are currently available from this group, the Digital Anatomist Browser, and a video disk featuring animations of the brain. The Browser is a front end to a multimedia server accessible on the Internet and has been used to teach neuroanatomy for 2 years. The Human Brain laserdisk contains a number of 3D animations of the brain from various perspectives and in various states of dissection. The model used for the animations was created by digitizing slices of a real brain and then rendering the resulting
data points on a high end graphics workstation. This same technique was used by Dr. Peter Ratiu of the Digital Anatomist group to create the animations in my Knee module.
Chapter 2

The Digital Dissector Design, Part I

2.1 Background

The idea behind the modules which I wrote comes from the Center for Advanced Biomedical Education (CABE) at Beth Israel Hospital in Boston and Wilson C. Hayes, Ph.D., a principal of CABE and a faculty member at MIT. Dr. Hayes is interested in bringing computers into the classroom for the purpose of teaching medical students, and one of his ideas is to use computers in dissection labs. It is hoped that students will find lab assignments easier and more beneficial if they are assisted through multimedia software. Based on this idea I decided to write software modules which will help students during dissection labs, possibly replacing the cumbersome dissection manual and anatomy atlas combination. The software I have written is being freely licensed to CABE in hopes that it will be expanded and used by future students.

2.1.1 Dissection Labs Now

During standard training, medical students are required to take an anatomy course that involves dissecting a human cadaver. As would be expected, the areas to be explored in lab are discussed beforehand in lecture. The students are then required to prepare for the lab by reading from a dissection manual\textsuperscript{1} which describes the steps and techniques to be used during dissection. This manual is extremely dense, primarily consisting of descriptions of the correct procedure for dissection and explanations about the structure of the relevant anatomical parts. There are only a few black and white illustrations to augment the dissector text, so to make up for this the dissection manual frequently refers to illustrations in an anatomy atlas. The atlas contains a large number of color drawings of

\textsuperscript{1} The dissection manual is commonly called the dissector, and I will use these two terms interchangeably
just about every part of the body and a small number of scans, x-rays, and MRI's. In most
cases the illustrations are accompanied by a few explanatory paragraphs. Students also
attend prosections during which dissection techniques are demonstrated on prepared
specimens. However, these prosections sometimes occur after the completion of the lab.

As one might guess, the actual laboratory procedure leaves a little to be desired. The
student begins a regional dissection by following the steps outlined in the dissection
manual. However, human cadavers vary enough that identifying anatomical parts is hardly
a trivial task. As a result, during the dissection many references must be made back and
forth between the atlas, manual, and specimen. The dissection is not easy even with the
help of the atlas, since the atlas does not show actual pictures of human anatomy, only
unnaturally colored illustrations where everything is labeled and shown without its
surroundings. In a real body, fluids, muscles, vessels, etc., hide interesting areas and
everything tends to be the same color. Students may find the dissection manual technically
sophisticated and hard to follow. They may also be forced to learn by trial and error as the
lab proceeds because they are not familiar with the area being dissected. Finally, students
may find the details and technique of the dissection elusive.

2.1.2 Improvements

The above scenario suggests a number of improvements. First, the dissection manual
and atlas could be combined into a single entity. Second, pictures of real anatomy could be
presented in addition to illustrations. Someone could demonstrate the proper steps and
techniques to the students prior to the dissection lab. Higher level material could be
distributed to augment the confusing passages of the dissection manual. Besides just
information on the dissection, the manual could describe common medical practices and
how they relate to the anatomy being dissected. Finally, the students could be given some
method to test and improve their knowledge before going into the lab.
By implementing these changes, the lab would be easier to carry out and more beneficial to the student. Because it is ideal for presenting the wide variety of information necessary to effect the changes above, it makes sense to use multimedia software as the vehicle of implementation.

2.2 Primary Design Goals

2.2.1 Identification of the Primary Design Goal

It was clear that my software should take the student through the dissection step by step, and with no less information than available in the dissection manual and atlas. Even if I only ended up combining the dissector and atlas, students might find the software preferable over the two books because of speed and efficiency considerations. At the same time, it was not clear that the benefits of combining the two manuals would compensate for any features lost as a result of moving everything to a computer (see section 7.2-7.4). However, since I originally expected to be able to test the first module in lab, my initial plan was to quickly develop a prototype that I could then improve. Quick development meant only a few features, so I determined that my first module would concentrate on the main objective of getting students through the lab.

2.2.2 Identification of the Typical User

Having identified my primary goal, I decided that it was time to figure out what other aspects I should concentrate on. It became apparent that I should identify who my users would be since this would influence a number of design decisions. This was simple enough since my user base (medical students) was defined in the statement of my primary goal. However, I needed to determine what these users had in regard to computer skill level and prior knowledge of the subject matter being taught.

Obviously I could not make any assumption about the computer expertise of the medical students. Simply from the facts that most children now have computers at home
and those that do not tend to be exposed numerous times from elementary school to college, I could conclude that a large majority of my users would have used a computer before. However, it would be unwise to base any part of my software design on this assumption since doing so might cripple anyone who had no experience to the point that they would be unable to finish the lab given only my software. As a result, I had to make sure that the software was as easy to use as possible.

As far as prior knowledge was concerned, in general I could not safely conclude that my users would be familiar with many aspects of dissection. On the other hand, I could assume prior knowledge for the first module because the anatomy class in which I hoped to test my software would have already been through a few labs by the time I was finished coding. I could therefore assume that my “testers” would have some knowledge of the common terms used in medicine, the dissector and atlas format, and standard dissection lab procedures. At the same time, if I ended up writing a complete Digital Dissector to be used in lab from day one, I could not allow these assumptions to influence my module design because during the first lab I would have students with no previous experience using my software. As a result, I decided it was better to assume little or no prior knowledge, at least in regard to standard dissection procedures and the dissector and atlas formats. This decision helped me clarify what types of information I wanted to include in the software.

2.3 Information Issues

One of the hardest things to decide when designing a multimedia program is what information should be made accessible to the user. For me, this was relatively easy since I assumed that I would be creating the functional equivalent of a dissector-atlas combination. This obviously meant that I needed to have dissection lab instructions along with illustrations and pictures. It was also important that the information I presented was
equivalent to that contained in standard lab manuals. While I could make additional
information available, it would be risky to use non-standard core information because
doing so might handicap my users and make it harder to analyze the benefits of my
software. I therefore decided to use text and pictures straight from Grant’s Dissector\(^1\) and
Grant’s Atlas of Anatomy.\(^2,3\)

Since I was assuming that my users would not have much prior knowledge, I thought it
would be a good idea to include background information and examples in the software.
This gave me the idea of using video clips of an actual dissection to demonstrate each step
of the lab. I also thought that videos or animations of anatomy like those seen on PBS
shows would be useful for demonstrating how the anatomical parts functioned. Together, I
felt that the dissector text, atlas pictures, animations, and video clips would be enough for
my prototype module.

2.4 Module Structure

Once I had identified what kind of information I wanted to include in the module, it
was necessary to choose a module structure. Typically, multimedia applications are very
unstructured, with a lot of interconnected nodes.\(^4\) However, I did not think that this sort of
structure would promote the kind of goal I wanted the software to accomplish. As a result,
I researched the factors that influence module structure. It turned out that there were a
number of issues that needed to be considered, but in the end, the obvious solution was the
correct one. In this section I present some of those issues.

---

1. Sauerland, Eberhardt: *Grant’s Dissector*; Williams and Wilkins, Baltimore 1991
3. I currently have no plans to sell or distribute my software, so this should not be a problem.
4. Jonassen & Grabinger, pp 12-14
2.4.2 Structural Implications of the Module Goal

First and foremost, the goal of the software should be considered when determining module structure. In my case, this factor alone was enough to convince me that my modules should have a relatively rigid and linear structure. The whole point the software was to help students complete a task, so allowing them to deviate significantly from the dissection procedure would only increase their chances of not finishing. Furthermore, the freedom of an unstructured module would mean that each student would be learning something different. This would be problematic for dissection labs since everyone is responsible for the same material. Therefore it seemed that I should channel all the students down the same path, providing them with the opportunity to access elaborations, helpful pictures, and videos, as long as they ended up at the right place. In a standard dissection lab, the anatomy atlas serves as a means of clarification, so it was logical to assume that atlas page accesses would comprise most of the elaboration.

---

1. Landow, p 101
2.4.3 Cognitive Overhead

A second issue to be considered regarding module structure is that of cognitive overhead.\(^1\) Cognitive overhead refers to the added mental demand that results from presenting the user with multiple courses of action. One advantage of textbooks is that the author can present material in the way that best promotes learning. In other words, authors hopefully have a good idea of how information should be structured in order to maximize its effect. In unstructured hypermedia, however, it is not possible to insure that the user makes the best decisions when choosing where to go next. For instance, the user may be pursuing a train of thought through the module but accidentally make a wrong decision regarding which link to follow next. This error may then cause the user to lose their train of thought. Alternatively, the path that the user follows may result in a wildly changing conceptual model because information is not gathered in a logical order. To prevent such mishaps it is necessary for the user to monitor their own mental state. The problem is that all this additional “cognitive overhead” may detract from the learning process. As one researcher says, “the richness of non-linear representation carries a risk of potential intellectual indigestion, loss of goal-directness, and cognitive entropy.”\(^3\) Because studies have shown that users with less prior knowledge are more susceptible to cognitive overhead and I was assuming that my users would be novices, I had an additional reason to worry about this problem.

Minimizing cognitive overhead had direct implications regarding how the modules should be structured. One obvious way to reduce cognitive overhead was to constrain the options available to the user. This implied fewer links between nodes. A second way to minimize it was to increase the level of organization within the module.\(^4\)

---

1. Jonassen & Grabinger, p 20
2. Mayes, Kibby, & Anderson, pp 236-237
3. Dede
4. Lowyck & Elen, p 141
discussion above, it should be apparent that cognitive overhead will be more of a factor in
large, highly interconnected hypermedia. Minimizing it therefore suggested that the linear
structure I had in mind would be better than the alternatives.

2.4.4 Conceptual/Module Structure

The third and final issue relating to structure involves the relationship between the
concepts that the author wishes to communicate to the user and the module structure. If the
author presents information directly to the user, some concepts may be “hidden” in the
structure of the hyper-document. For instance, the document’s structure might reflect a
conceptual hierarchy or links in the document might reflect connections between ideas. To
use a concrete example, imagine a hyper-document designed to teach students about the
phases of water. It begins with an ice cube and a puddle of water on the screen. Clicking
the block of ice activates a link that jumps to a picture of water in its ice crystal form. Thus
this link represents the idea that ice is composed of water molecules in a certain structure.
Clicking a water molecule in the ice crystal activates a link that jumps to an illustration of
how hydrogen and oxygen bond to form water, and this link represents the concept that
water is composed of hydrogen and oxygen.

On the other hand, the system designer may choose to teach concepts indirectly by
facilitating the completion of a task either inside or outside of the computer. In certain
circumstances such as my own, this is the only option, while in others it is simply an
alternative. From a structural standpoint, there is no reason for task completion modules to
be any more complicated than needed to achieve the task. Since no concepts need to be
represented in module links, the structure should be simple enough that the learning
process will not be impeded. Because the task I was attempting to facilitate involved
following a series of steps, it seemed clear enough that my module structure should be
primarily linear.¹
2.4.5 Structural Conclusion

As I had originally guessed, all factors indicated that my module structure should be simple and linear. As a result, I chose a model based on the idea that the user should progress sequentially through the dissection steps, digressing only to get clarification or help from a relevant atlas page, video, or picture. This structure could be imposed by connecting the dissector nodes together linearly with branching links to atlas pages (reflecting dissector references). The atlas pages should not have any links, however, thereby forcing the user to return back the main flow of the module after any digression.

![Figure 2.6: A Highly Structured, Linear Module](image)

2.4.7 Node Granularity

Having identified what kind of information I wanted to include and how I wanted to structure things, I now needed to decide how much information I would put in each node. By the time I was at this stage, I had already determined that my prototype module would focus on the larynx. The first subsection of the larynx chapter of the dissection manual provided an introduction to the larynx, describing its primary features. I decided that all of this information could be grouped together onto one page along with a picture of the entire

---

1. Silva, p 155
larynx. I decided to break up the rest of the larynx section into paragraphs and put one or two paragraphs on a node. Along with the text, I planned to include some relevant picture and the video clips demonstrating the steps explained in the paragraphs.

The atlas portion of the module was much easier to segment. The dissector refers to the anatomy atlas in terms of subsections, each which has a number like “8.32.” Individual subsections typically have a single picture and a few paragraphs of explanatory text. This seemed to match the format I planned to use for the dissector, so I decided to make each atlas subsection one node.

2.4.8 Module Layout

By now I had a clear idea of what I wanted the user to perceive as the structure of the module. First, I expected the software to start on a menu page from which every other page in the module could be accessed. Furthermore, every page in the module would have a simple way to return to the menu. The menu would therefore be the top level or root node of my module. The second level of the module would be the series of pages devoted to the actual dissection procedure. The third level would consist of the “elaborative material” including information from the anatomy atlas and other relevant sources. Hopefully I could make this structure apparent in the menu, and the user would figure it out after a few minutes of use.

![Figure 2.9: The Digital Dissector Structure](image-url)
2.5 Preliminary Objections

At first look, incorporating a dissection manual and anatomy atlas into a multimedia program seems like a good idea except for four things: cost, the hazards of having computers in the lab, development issues, and the fact that multimedia may not be good for this application. In this section I address all four problems but pay special attention to the last issue.

2.5.1 Cost

Cost is definitely a problem, at least if a large number of machines are needed. However, because of the fact that students take the quality of the learning environment into account when deciding where to go, schools need to computerize in order to stay competitive. More importantly, the price of multimedia-capable machines is on a steady decline thanks to vendor price wars and the fast pace of technology. It is also possible that in the long run, computerizing will prove to be more cost effective then buying textbooks and teaching “the old way.” If it turns out that the advantages of computerized instruction include significantly faster absorption rates and longer retention times, computers used in the classroom will pay for themselves in a very short amount of time.

2.5.2 Computers in the Lab

The problem of needing to use the software in lab is the most difficult to solve. The concern here is that if computers replace or augment the standard dissection manual in lab, students will transfer the grime from their specimens to the computer during the normal course of use. Fluids or flesh may get into the machine or fall onto the mouse pad, etc. If such events become common place, it will be hard to justify using an expensive piece of machinery rather than an easily replaceable and moderately resilient book.

Keeping things from falling into the computer and peripherals is prevented easily enough by encasing the keyboard and mouse in plastic membranes and by putting the CPU and monitor in special boxes. All of these items can be purchased for a reasonable
price. If something falls on the mouse pad, a roll of paper towels will do, as will a spare mouse pad. In fact, the mouse can be done away with entirely by using a touch screen monitor. Another alternative is to have students pair up into teams and have one team member interact with the computer while the other works. So while there is definitely the potential for trouble, a little care and common sense can prevent mishaps.

2.5.3 Maintenance

The third problem cited above relates to the maintenance of the software. New facts may be discovered, old information might become obsolete, the focus of the lab course might change, and other factors might influence the course in such a way that a change would need to be made in the software. Furthermore, when things need updating, the original developer will no longer be available for system maintenance so modifying the software will suddenly become a very real and difficult issue. However, with a little foresight on my own part as the designer, this problem can be easily prevented. Standard software engineering techniques such as modular design and full documentation can insure that any competent programmer will be able to modify the system whenever necessary.

2.5.4 Applicability of Multimedia in this Instance

Determining the utility of a multimedia program that helps students during dissection labs revolves around two questions. First, is it practical, and second, is multimedia suited for this particular task. The three points above address the practicality issue and if anything, I think the most difficult problem will be getting computers in the lab. On the other hand, the most important problem is establishing that multimedia is useful for highly structured applications such as directing lab work, since even if getting machines into lab is troublesome, computers can still be used to prepare for dissections. This problem basically comes down to a question of whether multimedia is useful for directed learning. In hypermedia literature, a number of researchers argue that multimedia is only useful for
undirected learning, but I believe a good argument exists for the other side as well. Rather than assuring the reader, though, I will present the argument.

Teaching may occur in a number of ways, and can be directed or undirected. Directed learning involves leading the student along a predetermined path, while undirected learning lets the student follow his own interests. Different subjects and goals may be better suited for a particular method\(^1\). Highly structured learning promotes uniformity of knowledge across students, but sometimes at the expense of better understanding. In conventional courses, the main tool of learning is the textbook. When using a textbook, students typically read information in the same order that it appears in the chapters, and as a result there is not likely to be much student directed browsing or searching. Therefore, the sequence of information in the book determines the conceptual model produced in the mind of the student.

Unlike textbooks, unstructured multimedia gives the student control over the order in which information is explored and allows for a wider variety of conceptual models. To illustrate, imagine two ways of teaching a subject: A subtopic is picked and thoroughly covered, then the next subtopic is covered, and so on. Alternatively, one aspect of the subject is picked and all subtopics are analyzed in terms of this aspect. Then another aspect is picked, and so on. Research has found that the first method focuses the student’s attention on how the information about individual topics is intertwined while the second method causes the student to concentrate on the similarities and differences between topics.\(^2\) Furthermore, evidence from these and other studies\(^3\) indicates that the first method is better suited to students with less prior knowledge about the subject because less mental context switching is required. Similarly, the latter method was found to be

---

1. Harland, p 89
2. Schnotz pp 81-98
3. Mandl, Schnotz, Picard, Henninger pp 74-75
more effective for students with greater prior knowledge. An unstructured multimedia program allows for either method, while a regular textbook forces the reader into one method or the other.

However, this undirected nature of multimedia may make it unsuitable for directed learning. If it is agreed that hypermedia documents are best suited to informal learning where the user follows interesting trains of thought through the software, we can conclude that what drives browsing and learning is user interest and curiosity. Therefore, as one researcher contends “Attempts to structure the information elements in particular ways, even if these attempts are meant to assist the user in understanding the information encountered, may instead do violence in some way to the user’s particular interest at that moment in time in the topic being explored. One should therefore be cautious of structure.” Based on this reasoning, it is possible to argue that conventional hypermedia is not very useful for highly structured learning tasks.

However, other scientists provide an answer to this claim, citing hypermedia as a good way to access elaborations on core information. Since the goal in a directed lesson is to provide the student with specific concepts and ideas, the structure of the hyper-document will necessarily be more rigid than in an undirected lesson. However, even for directed learning it is necessary to provide elaboration or explanations. Hypermedia is attractive for this kind of activity since it allows students to explore a relevant side branch when necessary, and to do so very efficiently.

From this observation we can conclude that to be useful for directed learning, software must be fairly linear, but can have a large number of short “offshoots.”

1. Duchastel, pp 140-141
2. Whalley, p 65
3. Duffy & Knuth, p 204
4. Duffy & Knuth, p 201
5. Streitz & Hannemann, p 415
logical flow of the program should be very rigid and constrained to a narrow path since this will prevent the user from getting side tracked or lost on an irrelevant branch. At the same time, a large number of short, unconnected digressions can be made available for students who need elaboration or examples. Thus, because the software allows instant access to elaborations while maintaining a directed approach, multimedia seems superior to books for directed learning as well. In fact, the structure of the module described exactly matches my own.
Chapter 3

The Development Environment

3.1 Hardware

Development was done on a Macintosh Quadra 850 with a 21” color monitor, 1 gigabyte hard drive, and 24 megabytes of RAM. While I did not need all of the disk space for the two modules I wrote, a large hard drive will be necessary to store all of the pictures, animations, etc., if development continues. The 24 megabytes of RAM was also more than I needed but was useful for complex image editing and multitasking. The most important aspects of the system from a hardware perspective turned out to be the color monitor and processor speed. Without color, the software quality would have suffered greatly and much of the detail in the images I used would have been lost. The processor speed was an important factor because the scripting language that I used to develop my modules was rather slow.

Because I needed to scan images from two books, a 4” black and white strip scanner was obtained for the first module. By the time I was developing the second module, a flatbed color scanner had been located. This made a tremendous amount of difference and second module’s pages are much more attractive. The animations I used in the second module were created on a Silicon Graphics workstation using custom software developed by Jeff Prothero, a researcher at the University of Washington.

3.2 Software: A.D.A.M.

Since my modules were intended to help students during dissection labs, it made sense to work within the A.D.A.M. software package. A.D.A.M. is an attempt at a virtual cadaver. The program begins with a human body in a central window, and the user can...
select the race, gender, and one of four views (anterior, posterior, lateral, medial). Clicking on an area of the body magnifies that region. The user can then “box cut” which strips off one layer of anatomy inside a user defined box or use a “scalpel” which strips off an oval shaped layer of anatomy wherever the user “cuts.” Since these methods usually turn out to be too slow, a third alternative is to strip off entire layers of anatomy using a slider on the side of the window. Clicking on an organ, bone, etc., produces a pop-up menu that displays the name of the selected body part. The menu sometimes contains links to histology (tissue samples) or radiology (x-rays) images, and can also access a list of links that the user can add to. A.D.A.M. also contains a rudimentary testing function and a few animations of common surgical techniques. However, it does not allow you to simulate a real dissection and is really only useful as an anatomy atlas. See Figure A.1 in Appendix A for a picture of the A.D.A.M. interface.

A.D.A.M. also has a project editor which allows developers to create their own multimedia projects using the Supercard scripting language. Developers can make their projects available to other A.D.A.M. users either by linking them to a particular anatomical part (via the link list described above) or through the Projects function which lets users open an arbitrary project. As a result, even though my modules were written “under A.D.A.M.” they are really only standard Supercard scripts.

### 3.3 Supercard

Supercard is a superset of the well-known Hypertalk scripting language. Hypertalk is a high level, english-like, interpreted language that is object oriented and event driven. In Hypercard, a program is referred to as a “Hypercard stack” since the metaphor used is that of a stack of index cards, where each node is a card. In Supercard, a program is a *project*, but nodes are still called *cards*. 
A Supercard project can be thought of as a hyperbook. Again, the nodes of this book are objects called cards, and the user is presented with one card of information at a time. Cards can contain text field, graphic, and button objects. Text fields can only contain text, buttons almost always have text based names\(^1\), and “graphics” can be arbitrarily shaped bits of graphics. Every object has a number and can be given a name, but different types of objects are addressed in different manners. For instance, the code to hide a text field is

\[
\text{hide cd field Help_text} \quad \text{or} \quad \text{hide cd field 3}
\]

while the code to hide a graphic is

\[
\text{hide cd grc Hyoid_bone} \quad \text{or} \quad \text{hide cd grc 7}
\]

The “cd” in these statements means that the function is being applied to a text field or graphic on the current card. Each object also has a set of properties specific to its type. For instance, text fields can be locked or unlocked, transparent or opaque, and so on. Furthermore each object can have a set of event driven scripts that control what it does in response to user or system activity.

### 3.3.1 Event Driven vs. Procedure Driven

Like most other Macintosh programs, Supercard is event driven rather than procedure driven. A procedure driven program executes code in a primarily linear fashion. In order to respond to user activity, a procedure driven program tests its input and branches to the appropriate place depending on input values. If no user activity occurs, the program must wait a bit and then check again. Event driven programs are more object oriented in nature and have segments of code that are executed only when a specific message is received from the operating system or run time environment. The Macintosh operating system is event driven, and sends messages to programs every time an “Apple event” occurs. These

---

\(^1\) If the programmer wishes to use a picture as a button name, the picture must be stored in a different file than the rest of the project.
messages tend to reflect user activity and include things like “mouseUp,” “mouseDown,” “openCard,” and “openProject.” Thus most Supercard object scripts are written to handle specific user actions. For instance, here is the script of a graphic that prints “HI!” on the screen when you press it:

```
  on mouseUp
    answer "HI"
  end mouseUp
```

The system will monitor for user activity, and if it detects mouse activity over the graphic, it will send messages to the graphic telling it what happened.

Most of the time, designers want their software to do something in response to user activity so event driven software is perfect. Because each object has its own script, the developer can specify what happens when a project or page is open or closed, when a button or graphic is clicked, and so on. Thus icons can be created by making a graphic with the desired picture and then writing a script that does the appropriate thing when the graphic is pressed. The graphic can also be made to change its picture to reflect the current state of affairs, thus acting both as a function and status icon.

In many cases it is desirable to let the user access an image or movie from a project, but Supercard does not provide a way to do this. Fortunately, the authors of A.D.A.M. included a mechanism to link PICT images, QuickTime movies (Apple’s digital movie format), other A.D.A.M. projects, LaserDisks, and Macromind Director movies to any project. When such a link is created, a special graphic is placed on the current page that displays the image or movie when it is clicked. By copying the script of these links into other objects it is possible to have any graphic or button display images or movies.

One drawback of using an interpreted language such as Supercard is that projects get slow as code is added. If each page does a substantial amount of housekeeping before
letting the user do anything, the watch icon will become a familiar sight (this icon is used to mean “wait a bit”). For this reason it is important to make scripts short and fast.

3.3.2 Development Environment: Conclusion

Because I was working on a Macintosh using Supercard, a number of user interface issues had already been decided for me. The Macintosh makes use of what is commonly called the “WIMP” interface (for “windows, icons, mouse, pull-down menus”). Unlike the software that runs on Intel based machines and unix workstations, it is therefore primarily mouse controlled. This means that pointing and clicking are usually all that is needed to get something done. For the sake of consistency, it made sense to preserve this in my software.
Chapter 4

Interface Design

4.1 Page Layout

In order to make things as simple and familiar as possible, it made sense to have a standard page layout. The simplest way to develop the layout was by experimenting on the computer, so at this point I actually started implementing the first module. I felt that to a certain extent, the module’s page layout should mimic a textbook page. This was mostly because I did not want the users to have to adapt to something totally foreign. Along these lines, I decided to keep the text together in one area and put any pictures below the text. For my first page I created a text field and typed in the introduction paragraph from the larynx section of the dissector. Below this text I pasted an image of the larynx that I had copied from ADAM’s virtual cadaver. I then sectioned off the top inch of the page, colored this region light blue, and typed in the page title. I hoped that making the page title distinct like this would make it easier for users to figure out where they were. I also left room at the bottom of the page for whatever page controls I ended up needing (see Figure A.3 of Appendix A for a picture of this page). Having come up with a satisfactory page layout, I now went about designing the rest of the user interface and coming up with the features that I thought the software should have.

4.2 Interface design

The user interface for my software clearly had to be easy to use. This was true for two reasons. First, my users were not going to be computer experts. This merely meant that I would have to make my software especially user friendly. Second, my software should avoid interfering with the completion of the dissection lab as much as possible. This meant
that moving around should be simple and straightforward.¹ I tried to guarantee that my software would be easy to use by implementing a small number of controls and making the interface consistent across the module. I also kept user input restricted to pointing and clicking.

4.2.1 Identification of necessary controls

In order to figure out what controls I needed to implement, I had to clarify what the user should be able to do with the software. Obviously there had to be links so the user could jump to referenced atlas pages, and once there, the user had to be able to get back to the main flow of the module. However, it did not seem like a good idea to use links to return from an atlas page. If an atlas page had one return link for every dissector page that referenced it, it would be possible for the user to choose the wrong return link and end up lost. Instead I decided to provide one “smart” link per atlas page that remembered where the user had come from. I was only going to allow the user to digress one page, so I could instantiate this as a “Previous Page” command.

In addition to links, I needed a small collection of commands that would allow the user to move through the module. The most obvious of these were the forward and backward commands which would be used to move through the linear sequence of dissector pages. I also decided it would be a good idea to include the previous page command on the pages in the main flow of the module. Finally, I would need commands that could be used to jump to the main menu and access help information. As far as implementation was concerned, I could use key-strokes, menus, buttons, or icons. Coherence to the WIMP design philosophy ruled out the first of these so I only needed to decide between the latter three. After thinking about these three options, I chose to use buttons rather than menus or icons, and this decision was based partly on efficiency and partly on utility.

¹ See Sutcliffe and vol F 67 of the NATO ASI series for more information on interface design.
4.2.2 Elimination of Menus

A standard way to shelter the user from input overload is by hiding functions in pop-up menus. Since most of the time the user is probably entering text or manipulating data (or in my case, browsing), it is not necessary to display every available command. Instead, functions are commonly divided up into related groups and these groups are accessible through menus. Buttons provide an alternative to menus when only a few functions are available or there is a set of frequently used commands. Accessing a function in a menu requires opening the menu and then selecting the function, whereas buttons only require a single click and are therefore more efficient to use.

Based on this observation, it was easy to decide against using menus. First of all, it would be senseless to hide the forward and backward commands in a menu. If users wanted to move back a few pages to see something, they would be forced to keep opening the menu and clicking the control. On the other hand, with a button or icon, this would only take a few clicks. There was no reason to go through the trouble of hiding the other three commands in a menu since it seemed clutter would not be a problem.

4.2.3 Buttons vs. Icons

In most cases icons are small pictures that, when clicked, invoke some function. They may also be used to indicate program status. Icons are especially useful for representing functions that are frequently used or take too long to describe with text. However, it is very common for an icon’s picture to poorly represent the function that the icon invokes. My decision to use buttons rather than icons was primarily based on this fact. Creating a button was just a matter of typing some text, and it was totally obvious what words I should use. On the other hand, it was not obvious what kind of graphic I should use for the “menu” or “previous page” commands. As a result, I decided to use buttons for everything.

1. Harland, p 84
4.3 The Menu and Help System

The only things left to design after the controls were the menu and help pages. After finishing these two, everything else would be a matter of implementation.

4.3.1 Help System

I hoped that my software would be intuitive enough that I would not need to supply much help information. At the same time I decided that the user should be able to get help from every page, whether this feature was context sensitive or not. In my original design, I hoped to have context sensitive help so that the user would not have to sift through irrelevant information. I planned on having at least three help pages, one for the menu, one for the dissector pages, and one for the atlas pages. The context sensitive feature would detect in which of these three locations the user had requested help and would display the appropriate screen. Until I had enough features implemented to merit three help pages, though, I decided to have one text based page with the expectations that it would evolve as I wrote code.

In addition to providing help pages I also thought it would be good to use the balloon help feature that Macintosh System 7 provides. Normal operation of a Macintosh with the System 7 operating system occurs with balloon help off, but if the user goes to the upper right corner of the screen, pressing on the balloon icon and selecting “Show Balloons” will turn it on. With balloon help on, some buttons and objects will display a bit of explanatory text when the mouse pointer is over them. This allows users to get an idea of what something will do without having to activate the help pages or use trial and error.

4.3.2 The Main Menu

Rather than getting fancy and implementing a graphical menu, I decided that the easiest thing to do would be to make a text outline of my module and then provide links from the outline to the actual module pages. I thought that this would be simple enough for the user to understand, and it would be relatively easy for me to modify if I wanted to
change the module. I wanted the user to be able to access the atlas pages as well, so I
decided to list them towards the bottom of the outline in a smaller font, or do something
similar to distinguish them from the main flow of the module. See Figure A.2 of Appendix
A for a picture of the Larynx module’s main menu.
Chapter 5

The Larynx Module

5.1 Module One, The Larynx

I originally planned on implementing a prototype module, testing it in an actual dissection lab, and then using the results to revise my design. Along these lines, Dr. Hayes found a dissection session which would be suitable for testing my software and we decided it would be a good idea to write a module for the larynx since that lab was towards the end of the term.

5.1.1 Dissector and Atlas Text

The first thing I did when I seriously began the implementation effort was type in all of the dissector text\footnote{If an effort is ever made to continue module development, I would highly recommend that the developer use a scanner with optical character recognition software rather than typing everything in by hand.}. I then broke it up into manageable pieces, created one blank page for each piece, and moved each to its own page. To keep the module looking professional, I then had to make sure that all the text boxes were in the same position on every page, otherwise when the user moved from one page to the next, the text would jump noticeably. I originally did this by carefully tweaking the box on each page, but learned that the easiest thing to do was to make one page with an empty text field and copy and paste this page a number of times, filling the text in afterwards. This procedure becomes especially useful when the page layout is more developed and contains buttons.

Next, I made one page for each atlas subsection referenced in the larynx text and entered the text from each subsection onto its individual page. In a few instances the subsection text was rather lengthy and took up a fair amount of the page. Because I wanted to put the subsection picture on the same page as the text, I could either decrease the text...
font, shrink the picture, or put the text in a scrolling text field. Separating the text and picture was not an option since the explanatory text would be useless without the picture it was describing. I tried using a smaller font, but the text became unreadable. I did not yet have access to a scanner, so I did not know what the pictures would look like when I shrink them down, but I assumed they would be acceptable. When I finally got a scanner, it turned out that the smaller the pictures were, the harder it was to identify their parts and the worse they looked. As a result, I ended up using scrolling text fields. I do not particularly like this solution, but was not able to come up with anything better.

Once I had entered all of the module's text, I was able to go through and give every page a title. For the dissector pages, I used the subsection names from the dissection manual. In cases where I had broken up a subsection, I used roman numerals to differentiate the pages. I then gave each page its Supercard name. This name is not seen by the user, and is only used for programming.

5.1.2 Two Important Issues

Before I get into the details of my coding, I should mention two things. First, when I started thinking about the design of my modules I realized that I would not be around after the end of the school year. I also knew that the CABE group wanted to get a full multimedia dissector up and running. This meant that while coding I would have to pay special attention to making my software expandable. In addition, I realized that things tend to change, information becomes outdated, new information is discovered, the focus of the dissection labs may change, and the dissector and atlas would be updated. This meant that my software would have to be easy to maintain and update. I fully intended to make expandability and ease of maintenance major priorities when coding, but with respect to the first module, one factor got in the way of this.
Before this project, I did not have any experience with Macintosh scripting languages and did not know that Supercard was a superset of Hypercard. When I began implementing the software I attempted to locate a Supercard manual, but all Supercard manuals were out of print and the local libraries did not have anything. As a result, I was forced to guess my way through the language until a quarter of the way through the second module when I accidentally discovered that a Hypercard manual would be adequate for my purposes. Before that time, I figured out how to do most of the things I wanted, but a number of commands eluded me, notably the ones that would have allowed me to write expandable and maintainable code. This will become apparent throughout the rest of this chapter.

5.1.3 The Module Controls

After laying out the module’s text fields, I began implementing the module controls. Thankfully, Supercard looks vaguely like English and the script editor has a menu that lists a large number of Supercard commands, so I was quickly able to get “Forward” and “Backward” buttons working despite the fact I did not have a reference manual. The most important line of the button scripts are nearly identical:

```
go next cd of this proj
```

for the “Forward” button, and the “Backward” button’s script is the same except instead of `next`, the script says `prev`. This line has the effect of displaying the next (or previous) card in the project.

Besides doing this, however, I also needed to keep track of the last page visited. The easiest way to do this would have been to put card names into a previous page variable by scripting something that said the equivalent of “put my name into the previous page variable.” The problem was that I knew how to get page names, but not without extra information (i.e. I knew how to get “card Larynx_Menu” but not “Larynx_Menu”). The
result was that if I tried to go to the card with the name that was returned, Supercard would signal an error.

To get around this, I just started hard coding page names into the button scripts. This meant that for the page named “Larynx_Menu,” I would have to code the equivalent of “put “Larynx_Menu” into the previous page variable”. This was not so difficult, but I had to do this for the forward, backward, menu, and previous page buttons. The result was that when I wanted to add a new page, I could copy and paste an old page but then I would have to go through just about every object’s scripts and update the page names. This turned out to be a much bigger problem when I started writing links.

Below are the scripts for the forward button from the page named “General_Remarks”. Again, the scripts for the backward button look almost exactly the same. The first script handles the page change while the second displays the balloon help information when balloon help is turned on. The lock/unlock lines prevent the window from being edited while I am displaying the new card.

**Forward Button Scripts**

```plaintext
on mouseUp
  global moduleView, curInk, adamShell, saveFlag, prevPage
  -- Declaration of global variables above.
  put "true" into saveFlag
  setWindow topWindow()
  send lockUp to proj adamShell
  -- Update the Previous Page variable
  put General_Remarks into prevPage
  -- Go to the next page
  go next cd of wd 1 of this proj
  unlock screen
end mouseUp

on mouseEnter
  set lockcursor to false
  FullBalloons "ShowDirect","Go forward one page."
end mouseEnter
```
The menu, help, and previous page buttons have scripts similar to the forward and backward buttons. They simply update any relevant system variables and then jump to the appropriate page. In the case of the menu button, I just hard coded in a jump to the menu page. For the previous page button, I jumped to the page name contained in the previous page variable. The help button was a little different in two ways. First, I did not want a visit to the help pages to affect the previous page command. For instance, if a user jumped to an atlas page then clicked help, upon exiting the help pages, clicking the previous page button should not take the user back to the help pages. As a result, I needed a separate variable to keep track of the page where the user invoked the help command. Secondly, I wanted the help pages to be able to demonstrate a number of the module features. As a result, I needed to initialize these features to a starting state. At the same time, I did not want to do this every time a help page was opened, just in case the user wanted to move among the help pages. The simplest way to do this was to use a boolean variable to indicate when I needed to reset the help page features (that is what the “H_1_H” in the script is).

Menu Button Scripts

```
on mouseUP
    global moduleView, curInk, adamShell, saveFlag, prevPage
    put "true" into saveFlag
    setWindow topWindow()
    send lockUp to proj adamShell
    -- Update the previous page variable
    put General_Remarks into prevPage
    -- Go to the main menu
    go cd Larynx_Menu of wd 1 of this proj
    unlock screen
end mouseUp

on mouseEnter
    set lockcursor to false
    FullBalloons "ShowDirect","Go to the Main Menu"
end mouseEnter
```
5.1.4 Link Implementation

After completing the page controls, I began implementing links. Links are primarily intended to take the user from a dissector page to an atlas page referenced in the dissector text. The dissector text cites atlas subsections by number, so I planned on linking the atlas pages through their subsection numbers. At first, I only wanted the links to do two simple
things: take the user to the right page, and change their appearance to let the user know that the page they pointed to had been visited. My primary influence was Mosaic, so I originally thought that the best way to represent a link would be to bold face or underlined the atlas subsections. The problem was that I could not figure out how to change the boldface or underlining once the link had been visited. My second idea was to put outline boxes around the appropriate words and have these boxes change color (or something similar) once their page had been visited. By hard coding page names into the link’s script, getting the link to jump to the right page was simple. Changing link colors was a little more complicated.

The whole point of changing a link’s color (if the page it jumps to has been visited) is to keep the user from getting lost or going to the same places over and over again. This effect should be preserved across pages so that if the user has already visited atlas page 8.56, regardless of what page she is on, any link jumping to atlas page 8.56 should not be its original color. This also means that the user can go to the main menu at any time and see every page that she has visited.

Ideally, to keep track of which pages had been visited, I would have used a single array with one boolean entry per page. However, I did not know how to make an array (due to lack of a manual). Even if I had, it became clear that I would have had to hard code the array because I could not get the correct page names I needed to index into it. I discovered how to implement arrays before completing the Larynx module, but by that time my code was operational and time constraints prevented me from going back and updating everything.

Instead of an array, I used one variable per page to keep track of whether or not a given page had been visited. Because I had to use hard coded page names, this meant that each link would have to be individually scripted. The link script I ended up writing works as
follows: when the user clicks on a link, the link puts the name of the current page into the previous page variable. It then updates the variable for the page it jumps to. Finally, it jumps to the right page. Below is the portion of code from a link that jumps to atlas page 8.69.

A Link Script

```plaintext
on mouseUp
  global moduleView, curInk, adamShell, saveFlag, prevPage, L8.69
  put "true" into saveFlag
  setWindow topWindow()
  -- Lock everything so the user can't mess stuff up and
  -- screen modifications happen at the same time
  send lockUp to proj adamShell
  -- We need to keep track of where we are, so we can get
  -- back using the previous page button
  put General_Remarks into prevPage
  -- We also need to keep track of which pages we've been
  -- to so we can change the link color
  put "true" into L8.69
  go cd 8.69 of wd 1 of this proj
  unlock screen
end mouseUp
```

Now that I could tell when a page had been visited, I needed to color its links appropriately. To do this I had to know the name of every link on the current page and where each of these links went. I could then test whether the destination of a given link had been visited, and if so, I could change its color. Because the state of a project is persistent even after the user quits, I also needed to reset link boxes to their “unvisited color” if their destination had not been visited. If I had not done this, when a user started up the module, link colors would reflect the state that the previous user had left them in (i.e. some visited and some not). Because I needed to toggle links between visited and unvisited states, I started using solid colored boxes instead of outlined boxes. I chose a dark red for the original “unvisited” color and a much lighter red/pink for the “visited” color. The link boxes were transparent, so I could place them over whatever text I wanted.
Below is a section of code from a page script that checks and updates information for a single link. In the first module, a portion of code similar to this needs to be written for every link on the page (not just for every atlas page referenced, but for every reference, duplicate or otherwise). Notice that the code does not refer to the link object by name, but by number. I ended up using numbers because I thought it would be easier to modify the page scripts if links were referenced by number, but this turned out to be a mistake.

```
-- If the user has been to page 8.69 then lighten
-- the link color to color 7, otherwise reset it to 19.
-- L8.69 is true if page 8.69 has been visted
-- The link box is object 13 (grc 13)
if L8.69 then set fillFore of grc 13 of this cd to 7
else set fillFore of grc 13 of this cd to 19
```

5.1.5 Atlas Pages

Around this point, I managed to acquire a strip scanner and I began scanning in the pictures from the dissector and atlas. Because of the scanner resolution, the labels on the picture were unreadable and in certain places it was impossible to make out the details of the image. Some images also had to be scanned in more than one strip. Therefore, turning the pictures into something useful required a significant amount of image manipulation. Luckily I had access to a powerful black and white image editor (Enhance) so this was relatively straightforward. Editing the pictures took a very long time, though, and was probably the most time consuming part of module development.

Once I had the pictures looking presentable, I placed them on their respective atlas pages. At this time it became clear I would have to make the atlas text reside in a scrolling text field. I then began placing control buttons at the bottom of the atlas pages. However, I did not use the complete set of controls, leaving off the forward and backward buttons that would have allowed the user to browse through the atlas pages. This was because I wanted
to prevent the user from digressing more than one page. Thus if a user jumped to an atlas page, the controls at the bottom would only consist of “Previous page,” “Menu,” and “Help.” Clicking previous page would return them back to the dissector page they had come from. See Figure A.5 of Appendix A for a picture of a Larynx module atlas page.

5.1.6 Help Mechanisms
Although I hoped that the software would be easy enough to use that instructions would not be necessary, it would have been irrational not to include some sort of help. Contrary to what I had expected, it turned out that two help pages were sufficient to cover all of the features of the software. In addition, there was no significant difference between the dissector pages and atlas pages. For these reasons I felt it was unnecessary to use context sensitive help. The help page layout I ended up using was simple, with a column of links and buttons on the left and explanatory text on the right. I made most of the links and buttons active so that the user could see an example of what they would do. I even provided a mechanism to reset all of the link colors back to their original dark red. As described above, I also made use of the balloon help mechanism provided by the operating system. See Figure A.6 for a picture of the primary Larynx module help page.

5.2 Additional features
After finishing the atlas pages, I now had completed enough of the module that it was usable. However, it was still borderline user unfriendly, so I decided to add a few more features.

5.2.1 Cursor Control
A frequently overlooked icon is the mouse pointer icon\(^1\), which, for Supercard is usually a hand or an I-beam. The I-beam is commonly used when entering or editing text, while the hand is used at just about every other time. By changing the mouse pointer, a

\(^1\) Oliveira, p 261
program can alert the user of status information or the fact that a special function is active. This technique is not used much outside of drawing programs even though it is a powerful way to give the user an idea of what will happen when the mouse button is pressed.

In order to maximize the likelihood that users would find my software easy to use, I had to provide them with as many clues about status as possible. As a result, I chose to implement a crude form of cursor control. In regard to the atlas links, I figured it would be clear enough that clicking on the red boxes would probably do something. However, to make sure, I decided to have the cursor change shape every time clicking the mouse button would do something. Most of the time, the mouse pointer icon is in the shape of a hand, so I thought that every time the mouse moved over a link or other area that would do something when clicked, I could change the pointer to an arrow. To do this I simply wrote two new scripts and inserted them into every clickable object. The first script changes the cursor to an arrow when the mouse pointer enters the object, and the second one changes it back to a hand when the pointer leaves:

```
on mouseEnter
    set lockCursor to "true"
    set cursor to arrow
end mouseEnter

on mouseLeave
    set lockCursor to "false"
    set cursor to hand
end mouseLeave
```

5.2.2 Anatomical Identification “Links” in the Dissector Text

As explained above, a standard page in the Larynx module consisted of some text and an illustration. To help connect the two parts together, I implemented a method for graphically identifying anatomical parts mentioned in the text. I placed transparent blue

1. Oliveira, p 261
boxes over the names of certain anatomical parts in the text, and clicking on these boxes would toggle thick circles or ovals around the appropriate anatomical parts in the picture.

Here is an excerpt from a blue box’s script:

```plaintext
-- 1. Show, Plain, Very Fast
-- GR_3_H: GR = General Remarks, 3 = graphic #, H = Hidden
on mouseUp
    global moduleView, curInk, adamShell, saveFlag, GR_3_H
    put "true" into saveFlag
    setWindow topWindow()
    send lockUp to proj adamShell
    -- Test if the circle is hidden. If so, show it, otherwise hide it. Update the variable that records the circle's status.
    if GR_3_H then
        put "false" into GR_3_H
        show cd_grc GR_3
    else
        put "true" into GR_3_H
        hide cd_grc GR_3
    end if
    unlock screen with Plain Very Fast
    set cursor to arrow
end mouseUp
```

5.2.3 Anatomical Identification and Link Menus in Dissector Illustrations

The A.D.A.M. software allows the user to identify areas of anatomy by holding down the mouse button over the region in question. A pop-up menu will then appear which displays the name of the selected part. In most circumstances, this pop-up menu will also allow the user to access histology and radiology images. Since this is a much better way to identify anatomy than using circles activated from the text, I decided to implement it as well.

Initially, I implemented this using a collection of completely transparent objects that would display the relevant pop-up menu when clicked. I first created an opaque object in the shape of the region I wanted the pop-up menu to activate for, then wrote the script
which activated the menu and displayed the appropriate information. Next I placed the object over the proper area of the dissector illustration and made it completely clear.

There were two problems with this technique. First, if I needed to move the picture for any reason, I had to move all of these small transparent objects. However, it was rather difficult to position them because they were completely clear. The second problem was that covering all the relevant anatomical parts in a given picture required too many of these objects.

My solution was to create a zone sensitive script for the original picture, so no additional objects were needed. This script works as follows: When the user clicks the mouse button inside the picture, the script checks to see where the click occurred. Using a series of if-then statements, the script figures out what anatomical part has been clicked. It then calls a procedure to create the appropriate pop-up menu. Unfortunately, the sensitive zones must be rectangular. This is not a serious problem because most areas actually are vaguely rectangular, and those that are not can be broken down into two or three regions.

Because I wanted to let future programmers expand the software as much as possible, I made a mechanism to easily change the information displayed in the pop-up menus. At the beginning of the object's script, I created a list of anatomical part names and images that should be listed in the pop-up menu. When the pop-up menu procedure is called, it searches through this list for the right entries and uses them to create the menu. Thus adding more options to the pop-up menu is as simple as adding more lines to the list. It is even possible to modify the software so menu items activate links to other pages or other projects (right now they only display pictures). This technique is a little bit on the slow side, but is relatively unnoticeable to the user as long as there are not too many things in the list. If it becomes necessary to add a large number of entries, the list should be
reordered with the most frequently accessed parts first (probably the largest parts). Below is a sample list.

Thyroid Cartilage,2,Histology,Radiology
PICT,Macintosh HD:A.D.A.M. Folder:HS8.08,?,?,1/17/94
PICT,Macintosh HD:A.D.A.M. Folder:ML.05,?,?,1/17/94
Superior pharyngeal constrictor muscle,1,Histology
PICT,Macintosh HD:A.D.A.M. Folder:HS8.21,?,?,1/17/94
Hyoid bone,1,Histology
END

The first line indicates that for the thyroid cartilage, there are 2 menu entries and they are histology and radiology. The name and location of the actual pictures that should be displayed are listed on the next two lines. This list only has data for three regions (thyroid cartilage, superior pharyngeal constrictor, and hyoid bone). Since the actual script which creates and displays the menu is somewhat complicated, I will not list it here. The reader can find it in Appendix B.5.

5.2.4 Linked PICTS

Besides the illustrations from the dissection manual and atlas, I wanted to include pictures of actual larynxes, x-rays, CAT scans, and the like. Because A.D.A.M. provides a way to link in PICT format images, I decided this was the easiest and best way to go. The scanner that I had access to during the development of the Larynx module left a lot to be desired, so I ended up borrowing most of my images from A.D.A.M. (A.D.A.M. has a collection of images accessible through the main program. I tracked these down and linked them to my module.) The PICT viewer provided with A.D.A.M. needs a little work but has some nice features. Initially the PICT image is displayed in a small window which can be enlarged by clicking an "Enlarge" button. While viewing the enlarged version, the user can click the mouse button and a 3" circle will be enlarged around the mouse pointer.
Unfortunately, because the PICT files have relatively low resolution, the zoomed area looks very grainy (see Figure A.13 for a picture of the PICT viewer).

Normally when you link a PICT through A.D.A.M., A.D.A.M. creates a special button for you. However, I found that I wanted to let the user access pictures through various means including graphics and pop-up menus. The simple solution was to copy the script of this button and paste it into objects that I wanted to display PICTs.

5.2.5 Student Use Tracking

In addition to helping students complete the dissection lab, I thought it would be a good idea to help teachers get a better idea of how their students were doing. In the Larynx module I was not overly concerned with this, but wanted to experiment and make sure that I could come up with some idea of how to do this. I therefore implemented a method for monitoring where students spent their time in the module.

When the module is started, the first thing that happens is the user gets asked for their name. Throughout the time that the student uses the software, I keep track of how long they spend on each page. Revisiting a page will not reset this amount but will add to it. When the user has finished using the software and closes the project, a script gets activated that opens a log file and records the user’s name and how long they were at each page. This file can then be viewed with a word processing program. Because this feature was only used for testing purposes in the Larynx module, I will not explain the code (although it is listed in Appendix B.1). An explanation of a better usage tracking mechanism is given in the next chapter.

5.3 Problems with the Larynx module

5.3.1 Object Expansion problems

After I finished writing the basic features, I tested the software myself and had students and staff from the Orthopaedic Biomechanics Laboratory at BIH test it for me.
The first two people to test my code noticed some strange behavior when they tried to use certain atlas links. Some links would not change color correctly, and others would report error messages.

While developing the Larynx module I had a number of dummy graphic objects that were supposed to be links to QuickTime movies. However, as I will describe below, I ended up not having any movies at all. Therefore, before having other people test my code, I went through and deleted all of these dummy links. It turned out that the problem my testers found was caused by A.D.A.M. renumbering my graphic objects when I deleted the dummy QuickTime links. As described in section 5.1.4, I was controlling the color of link objects by number, and I had hard coded these numbers in. On the pages where I deleted the dummy links, A.D.A.M. automatically renumbered everything and what used to be graphic 12, for example, was now graphic 10. This meant that sending a message to the last graphic would result in an error, because that number no longer existed. The links that had changed number would now color for the wrong page. While this was easy enough to fix by changing the hard coded numbers, it demonstrated a basic flaw in the technique I used to reference objects.

5.3.2 Page expansion problems.
Adding or deleting pages caused similar problems to those produced by adding or deleting graphic objects. This was also because of the name problem described above. I needed to keep a number of page specific variables:

1. The last page visited (for the “Previous Page” button).
2. The page that the user pressed the “Help” button on (I did not want to have the “Previous Page” button jump to back to the help pages after the user had just visited them, so I needed to store the departure page in another variable so the user could get back to where they had left from).
3. A list of all the pages that have ever been visited (so I could change link colors appropriately).
4. A list that kept track of how long a user had spent on every page (for the usage tracking feature).
Each "Forward," "Backward," "Menu," and "Previous Page," button had to be individually coded to insert the proper page name into the previous page variable. Each "Help" button had to be individually coded to insert the proper page name into the help page variable.

Since I did not know how to make an array, I had to keep two variables for each page. The first was used to determine whether or not the user had visited the page, and the second kept a running total of how long the user had spent there. Thus every time I added a page, I would have to spend a significant amount of time updating all the necessary scripts. As a result, the Larynx module was not easily expandable.

5.3.3 Control Problems

After getting a little feedback from testers, I ended up adding a "Print" button to the bottom of every page. Unfortunately, transparent objects print opaquely. Since I use a large number of these objects, printouts do not reflect what the user sees on the screen.

I also ended up adding a "Quit" button because of a loophole in the way that a user can quit the module. In order for the usage tracking feature to work, the project must be able to detect when it is being closed. If the user selects “Close Project” under the file menu at the top of the screen, everything works fine. However, if the user selects “Quit” under the file menu (this exits A.D.A.M.), the module will not be sent any indication that the project is being closed and the usage tracking mechanism will not work. In order to prevent this, I placed a large colored “QUIT” button which sends the proper messages to the project at the bottom of the main menu.

Unfortunately, by the time I had added the “Print” button, there was not enough room at the bottom of the page to comfortably fit all the controls. As a result, I changed the "Backward" and "Forward" buttons to "<-" and "->" respectively. The meanings of these
buttons seemed to be clear enough, but making them smaller de-emphasized their importance (since I wanted progress to be primarily linear).

5.3.4 Lack of Color Pictures.
The most notable thing about the Larynx module was the lack of color pictures. Fortunately I was able to use a color scanner for the second module, so this was no longer a problem. The color pictures that I do have in the Larynx module were captured from the A.D.A.M. “virtual cadaver.”

5.3.5 Circling Anatomy Was Not That Effective
Originally, I had placed blue “links” over words in the dissector text that would cause a circle to appear around certain anatomical parts. At the suggestion of initial testers, I changed this so that for black and white illustrations, the appropriate area of the picture suddenly becomes colored. The effects of this are much more dramatic and eye catching (see Figure A.4 for a picture of this). However, I was unable to come up with a good way to do the same thing with color pictures. I tried three solutions: 1) making the colors in the appropriate region especially bright, 2) dimming everything other than the relevant area, 3) blinking the appropriate area. Brightening the area turned out to be almost unnoticeable, while dimming the other areas turned out to be too complex. To dim the areas, I had to create completely transparent objects that could turn transparent grey on demand. However, too many of these objects were required to allow even a small number of anatomical parts to be highlighted. Blinking the area was also troublesome, since I either had to have a bunch of individual objects that made up the complete picture, or use a number of transparent objects that turned opaque on command. Probably the best solution is to do away with these boxes completely and display pop-up menus when the user clicks on the picture.
5.3.6 Tracking Mechanism Not Very Useful
I did not expect to implement an extremely useful tracking mechanism. As it turns out, knowing how long a user spends on a specific page is not all that useful, especially if the user leaves the terminal to do something else. Instead, it would be better to keep track of what the user does since repeated actions may indicate confusion as might frequent trips to the help page. I wrote a much more effective tracking mechanism for the second module.

5.3.7 Previous Page vs. Backtracking
I imagined that the “Previous Page” button would primarily be used to return from brief digressions to an atlas page and therefore did not need to have a memory greater than one page. However, my testers often tried to use it like the backtracking mechanisms for other hypermedia programs (e.g. Mosaic) which keep taking you back the way you came. This was fixed in the second module once I had figured out how to make an array and reference card names.

5.3.8 Getting Lost
As hard as I tried to keep things simple, getting lost still was a problem. I think that part of this was because the textual menu was somewhat misleading, and did not place enough emphasis on the linear structure that I had hoped to suggest. I also had a request from Dr. Hayes for something that would show students where they were in relation to the dissector text. The first problem will hopefully be fixed by an improved menu layout and the new backtrack button which should get users back to a familiar place. In the second module I also implemented a feature that shows the users where they are in relation to the actual dissection manual.

5.3.9 Text too Dense
On numerous pages of the Larynx module, the text is rather lengthy. I attempted to fix this in the second module by breaking the text into smaller chunks and having more pictures per page.
5.3.10 Too Much Mouse Movement

More of an annoyance than anything else was the fact that the module controls were at the bottom of the page, but most of the atlas and picture links were at the top of the page. This meant that users had to constantly move the mouse up and down the page, depending on what they were doing. Not only did this get tiring, it also made interacting with the software take that much longer. A major goal of the software was efficiency, so I made sure to fix this in the second module.

5.3.11 Lack of QuickTime Movies

I felt strongly that students would have an easier time in lab if they were able to see the lab demonstrated before doing it themselves. As a result I wanted to include video clips of actual dissections. I also wanted to include animations of anatomy so that students could get a better feel for how things worked and what things looked like before beginning the lab. Finally, I thought it would be relatively easy to include video clips of anatomy in action, such as the vocal chords moving, etc. (which just about everyone has seen on PBS). I discovered that I was wrong.

I originally planned on finding video tapes of dissections, and then converting them to QuickTime\(^1\) movies via a video grabber board. I began searching for video tapes at Beth Israel Hospital, but this proved to be a dead end, as did the Harvard medical school and Harvard medical library. I posted a question regarding where I could find video tapes on the Internet, and received numerous replies suggesting video production houses. However, when I called these places, it turned out that rental fees were $75 a day, much more than I was willing to pay. I also looked into video taping a dissection myself, but it turned out that this was also too expensive. As a result I was not able to include any animations or video clips in the Larynx module.

---

1. QuickTime is Apple’s digital movie format for computers. It works very well given the available technology.
This story has a happy ending, however, thanks to Dr. Peter Ratiu, an MD from the University of Washington. Dr. Ratiu responded to my Internet post and we exchanged a number of letters. It turned out that he was involved in the Digital Anatomist project and was creating 3D animations of the knee. He said that he would be happy to see his work go to good use, and I was welcome to use his animations. As a result, I decided to write the second module on the knee specifically because I knew that I could get animations. In fact, Dr. Ratiu was able to make a number of animations that I specifically requested and I will describe these in the next chapter.

5.3.12 Development Took Longer Than Expected
A practical problem that I encountered was that development did not proceed as quickly as I had hoped. As a result I was not able to finish the Larynx module in time to have it tested in an actual dissection lab. I checked for other dissection labs at MIT and Harvard, but did not find anything. As a result, I decided that it was probably better to concentrate on fixing what I knew was wrong with the Larynx module and let someone else determine if the software could actually be useful in lab. The focus of the second module is therefore on improving the Larynx module’s code and adding features.

5.3.13 Poor Coding Due to Lack of Manual
As described above, my attempts to acquire a Supercard manual were fruitless. The effect of this frustrating episode was that much of the code in the Larynx module was suboptimal because I did not have a manual, had never programmed with Supercard before, and therefore had to guess how to do almost everything. As a result, the code I wrote was often inefficient but worked, which was better than nothing. I noticed the relation between Supercard and Hypercard early enough during the development of the second module that I was able to rewrite most of my code and avoid the problems I had in the first module.
Chapter 6

The Knee Module

6.1 Introduction

After implementing and testing the Larynx module I was ready to begin work on the second module. Because I knew that I could get animations on the knee from Dr. Peter Ratiu who was working on the Digital Anatomist project at the University of Washington, I chose the knee as the subject of my next effort.

6.2 Design

Whereas the first module was more to establish proof of concept, the Knee module was designed from the outset to follow good educational multimedia guidelines. Overall, the format of the Knee module is very similar to the Larynx module. The main differences are in implementation details and module content. I concentrated on producing a foundation to achieve all of the design goals that I originally had in mind, including:

1. Helping students prepare for the dissection lab
2. Helping students complete the dissection lab
3. Presenting information regarding common injuries and how to repair them
4. Feature improvements that make the software a better aid to students as well as teachers
5. Ease of expandability and maintenance

6.2.1 Design: Module Layout

Once again, the primary goal of the module was to get the students through the dissection. However, I extended the main flow of the module to include three more sections in addition to the dissector and atlas pages. After the main menu, the first section of the module is comprised of a small group of pages which provide a brief overview of
the bones, muscles, tendons, etc., of the knee. This section is intended to supply students with a basic introduction to the anatomy of the knee. The second section is comprised of the dissection pages. The third section, called “Common Procedures,” illustrates a few knee injuries and surgical techniques. The last section of the main flow of the module consists of a quiz facility that students can use to test their knowledge. The atlas page arrangement is the same as in the Larynx module.

6.2.2 Design: Page Layout

I rearranged the page layout in the Knee module so that interacting with the software would be more efficient. The first thing that I took care of was moving the text and pictures so that links would be closer to the page controls at bottom of the page. This would prevent the user from constantly having to move back and forth between the top and bottom of the page. I also tried to increase the number of pictures per page and decrease the text density in order to make pages more readable. Finally, the buttons were rearranged to emphasis the importance of the forward and backward controls.

6.2.3 Knee Module Controls

As in the Larynx module, I needed forward, backward, menu, and help buttons. However, I wanted to minimize the area taken up by the module controls while re-emphasizing the importance of the forward and backward page controls. To do this, I drew two icons to use in place of the “<-” and “->” buttons. I got rid of the print button, both for the sake of space and because the print mechanism did not seem to be working. In place of the previous page button, I used a “Backtrack” button. I also added an “Annotate” button which toggles user editable text fields. As before, I placed a “QUIT” button prominently on the main menu page where users start.

The implementation of the page controls changed slightly because I realized it was better to have variables updated by page scripts than by button scripts. In the Larynx module, the forward, backward, help, menu, and previous page buttons were all
responsible for updating the previous page variable. In the Knee module, I replaced the previous page variable with a backtracking stack. To get the backtracking feature to work correctly, I had to keep track of every place the user had been. I therefore implemented a stack of page names, where the top name is the current page. Every time the user visits a new page, its name must be pushed onto the stack. Then backtracking simply involves popping two pages off the stack and going to the second one (since the first name is the current page). Repeated clicking of the backtrack button will pop successive pages off the stack. Since page scripts could take care of updating the stack, it was not necessary for the control buttons to do anything other than their primary function. The script for the backtracking button is in Appendix C.5.

6.3 Knee Module Features

6.3.1 Color Pictures

One of the most noticeable difference between the Knee module and Larynx module is the abundance of color pictures. Once again, I scanned pictures from the anatomy atlas, but because I had a color scanner and better image editing tools (i.e. Adobe Photoshop), the Knee module images are much nicer. The limiting factor of image quality is now Supercard, which does not seem to be able to handle anything more than 8 bit colors.

6.3.2 The Introduction and Common Procedures Pages

The introduction pages of the Knee module present very basic descriptions of different aspects of the knee through the use of labeled pictures and 3D animations. The “common procedures” pages have pictures illustrating some common injuries and the methods used to repair them. Each page also has an animation of the technique described. On a few pages, I also include video clips from actual surgeries. These clips were given to me by Dr. Ratiu. See Figure A.7 and Figure A.9 in Appendix A for pictures of the introduction and common procedures pages.
6.3.3 Animations

Possibly the second most notable feature of the Knee module is the presence of a number of animations. These animations include:

- The common procedure animations
- Bones of the Knee
- Cartilage of the Knee
- Menisci of the Knee
- Total Knee (bones, tendons, muscles, vessels, and nerves)

The common procedure animations were created by the authors of A.D.A.M. using Macromind Director and Supercard. They are really only recordings of Supercard objects interacting on the screen. In the future, I think it would be possible to use this technique to make animations of dissection procedures, although such animations may not be as useful as video clips of real dissections. A picture of one of these animations is in Figure A.16 of Appendix A.

The other four animations were created by Dr. Peter Ratiu. They show specific parts of the knee rotating in three dimensions so students can get a better idea of what the parts look like and how they fit together. A picture taken during the Bones of the Knee animation is in Figure A.14. This excerpt from one of Dr. Ratiu’s letters describes how he created one of the stunning 3D animated models that I have included in the software:

“...This particular model has been done from color slides taken from a fresh frozen specimen sectioned at 1 mm. Every structure has been digitized separately; the resurfacing has been done on Silicon Graphics, with home-made software, called Skandha (Jeff Prothero). The animation is recorded on the SGI as a sequence of RGB files. I can use any type of image source that gives me serial sections of a 3-D objects.”

In addition to these animations, Dr. Ratiu also gave me a number of “animated scans” produced by taking consecutive MRI scans of the knee and then putting them together in a
QuickTime movie. Playing the movie gives the effect of moving through the knee. I have three scans, one each for the axial, coronal, and sagittal planes. A picture of one of these movies is also in Figure A.15.

6.3.4 Annotation Mechanism

While reviewing literature on educational multimedia, I noticed that students frequently wanted a way to highlight or annotate software\(^1\) so I decided it would be a good idea to add this feature to the Knee module. I attempted to write something that would allow annotation of any part of a page, but I could not get it to work correctly. I originally copied the current page and pasted a user editable version on the screen and then activated the text tool so the user could type anywhere on the page\(^2\). However, I could not figure out how to get the mouse pointer to toggle between the text tool and the browsing tool once the user had finished annotating. As a result, the current implementation of the annotate button simply calls up a scrolling text field that the user can type in. Clicking the “Annotate” button again hides the text field. One drawback of this method is that annotation text fields needs to be placed individually on every page so that they are not covering up any information that the user may want to view while annotating. Since the literature seems to indicate students like to view each others’ annotations, I made one universal text field per page. It would be straightforward to change this so that each user had their own personal annotation field that no one else could read.

6.3.5 Dissection Manual Locator

Since students may not want to give up their dissection manuals right away, instead using them in conjunction with the software, it makes sense to indicate where in the actual dissection manual the text from a given module page comes from. To do this, I scanned in the dissector pages, then shrunk them down and made graphic objects out of them. On

---

1. Legget, Schnase, Kacmar pp 33-34
2. Supercard allows the mouse to function as one of a variety of tools. The default is the browse tool which is used to click on things, while the text tool is used for typing text.
each miniature dissector page I highlighted the area where the text of the current module page came from. I put this miniature page next to the module text. Thus, on a specific dissector page in the Knee module, the text in the window is the same as the text in the highlighted block of the miniature dissection manual. This fact allows users to quickly compare where they are in the software to where they are in the actual manual. In the future, it may be desirable to make these minipages toggle on and off (see Figure A.8 for a picture of a Knee module dissector page with a minipage).

6.3.6 Quiz Facility

I added a simple quizzing facility at the end of the Knee module so that users could get some idea of their level of knowledge. A.D.A.M.’s testing facility is better than my own because it names a specific part of anatomy and then asks the user to locate it in the “virtual cadaver.” My facility, on the other hand, works the other way around, asking a question graphically and then expecting a textual answer. I established a quiz file format so that it would be easy to expand the quiz in the future.

Right now, my quiz facility works by displaying up to two graphic objects, asking a question, and then comparing the user’s answer to up to two expected responses. For instance, the first quiz I implemented displays the bones of the knee and an arrow pointing to one of the bones. It then asks the user to name the indicated bone. A picture of this is in Figure A.11. If the user types in the correct name, either in all lower case or with the first letter capitalized (i.e. “femur” or “Femur”), the computer will display the message “Correct!” and will increment the user’s score by one. Entering any form of “quit” will stop the quiz, and clicking the cancel button or pressing return by itself will skip the current question. At the end of the quiz, the computer displays the user’s score. I have not implemented a mechanism for saving these scores, but this would be extremely easy to do.
6.3.7 Improved Usage Tracking Mechanism

In the first module I determined that it was not that hard to create a usage tracking mechanism. In the second module, I implemented a tracking mechanism that is actually useful. A number of tracking techniques are described in the literature\(^1\), with the most active systems logging every action of the user (i.e. tracking occurs at the Apple event level). I decided that this was overkill, and instead thought that it would be enough to record what the student did at the page level.

Once again, when the module is started, the first thing that happens is the user is asked for his or her name. Entering nothing or clicking cancel will turn off the usage tracking feature. Every time the user visits a page, a timer is started. When the user leaves the page, the page name is inserted into an array along with the time spent there. When the user quits out of the module (by closing the project or clicking the quit button), a file is created whose name is the current date concatenated with the user’s name. On the first line, the file contains the user’s name and total time spend in the module. The following lines list page names in the order that they were visited along with the amount of time the user spent on each visit. The information is tab delimited so that the file can be opened and analyzed in a standard spreadsheet program. Below is sample output from a tracking file.

```
John Smith   260
Knee_Menu    4
K_Bones      10
Help         30
K_Bones      5
K_Menisci    20
8.67         10
K_Menisci    4
...          
```

Because the array can be modified by any object, it would be very easy to keep track of all mouse activity. A programmer would only need to add a line to each button or clickable

---

1. MacLeod 24-31
graphic that would insert the object’s name (followed by a carriage return) into the tracking array whenever the object was pressed. By writing a script that triggered whenever the input was idle, it would also be possible to keep track of how much of each session was spent interacting with the software (as opposed to reading).

6.3.8 Programmers Page

During the development process, I did not like having to exit or restart the module every time I needed to reset a system variable or view a file. As a result, I made a special “programmer’s page” that has a number of useful buttons and a text field for input and output. There are about ten buttons which reset module variables, print variable contents, allow the programmer to view usage log files, and so on. The scripts of these buttons are listed in Appendix C, and a picture of the programmers page is in Figure A.12.

Since the A.D.A.M. interface provides a way to access every page in a module, it is possible for a student to visit the programmer’s page even though it is not listed in the menu and is out of the main flow of the module. As it would be dangerous to let students use the buttons on the programmer’s page, a large opaque object hides everything when the page is opened. A dialogue box then asks the user if they really meant to visit the programmer’s page. If so, entering the proper password (dana779) will reveal the buttons. Otherwise, the user is transported to the main menu.

Possibly the most useful button on the programmer’s page allows the programmer to update page or link scripts anywhere in the module. Since the code for most pages is the same, any change that needs to be made for one page will probably affect many pages. The same thing is true of links. Rather than going through and fixing everything by hand, the “update” button allows the programmer automate script edits. When this button is clicked, it first determines whether the programmer wants to update page scripts or link scripts. It then confirms that the replacement script has been placed in the programmer’s page text.
field. Next it asks the programmer for the range of pages that need to be updated. It then iterates through every page in the specified range, replacing the old page or link scripts with whatever is in the programmer's page text field. One important thing to note is that while this button is very helpful and powerful, it is also dangerous. It is extremely important that the replacement script is tested and the proper page ranges are specified. For instance, updating the programmers page with the script from the pages in the main flow will break it. Fortunately the actions of this button can be undone by using the "revert" command under the file menu, but all changes to the module since the last save will be lost. The code for this button is in Appendix C.11.

6.4 Code Improvements

6.4.1 Code Abstraction

The least noticeable but most important difference between the Larynx module and the Knee module is the quality of the code. The improvement is primarily due to manner in which object references are handled. As described in the previous chapter, the Larynx module scripts had to be hard coded with object names. The result of this was that whenever a module change was made, every effected object would have to be individually updated, and an update function like the one described in the previous section was not possible. The fact that I was unable to use arrays also meant that each piece of information that I wanted to store required its own variable.

These problems no longer exist in the Knee module. Now the only difference between objects with the same function (such as two pages or two "Menu" buttons) is the objects' names. This is because I figured out how to reference objects by name, even without doing so explicitly. I also discovered how to make arrays and lists. The fact that I could now use object names and arrays resulted in a number of improvements.
6.4.2 Link code improvements

I ended up doing two revisions of the atlas link scripts. The first rewrite made it possible for every link object to have the same script, the only difference between links being their names. This was possible because the new script said the equivalent of “go to the page with the same name as me,” and I simply named each link object after the page I wanted it to jump to. With this change, it did not matter if I added or deleted pages to the module, since page names would stay constant regardless of any other modifications I might make.

Using a similar idea, I rewrote the part of the page script that takes care of changing a link’s color if it has been visited. Rather than needing one variable for every page, I used a single array to keep track of where the user had been. The new code tests if a page has been visited by passing a special function the name of the page in question. This function calculates a page number given a page name, and uses the page number to index into the “visited” array. Because of the way this function works, a developer can add or delete a page and the next time the module is started, the array will automatically resize and link coloring will work correctly.

To keep from explicitly coding which objects on a page were links, I modified things so that the code would iterate through every object and look at each object’s name. Initially my code required that only link objects have names since all non link objects would return a name beginning with the word “card” (objects without names return information in the format “card ## graphic ##.”) However, I realized that in the future it might become necessary to give other objects names, such as in the case of my quiz page where I show and hide pictures according to their name. As a result, I did a second rewrite which requires that the name of a link object start with the word “link” and the second word of the name is the actual page name it jumps to. To implement this, the link scripts
had to be modified to say the equivalent of “go to the page with the same name as the second word of my name.” In the current implementation, this only works if page names are one word long. This is not really a problem since page names are not used except in my code (they are not the page title displayed on the screen). I imagine this would be easy to fix if necessary, but in the mean time I do not see any reason why page names need to be more than one word long.

6.4.3 Page Control Code Improvement

In the Larynx module, every page control button had to explicitly put the name of the current page into the previous page variable. In the Knee module, the backtracking stack replaced the previous page variable. I also moved the update function out of the buttons and into the page script, so rather than needing each button to update the stack, the current page pushes itself onto the stack as soon as it was opened. There was no need to do this explicitly either. Instead of saying “push the name “Knee Menu” onto the backtrack stack” the script can now say “push the name of this card onto the backtrack stack” without caring what page it is on. This means that the page control buttons do not contain any page specific information, and a button copied from one page can be pasted onto any other page and still work.

6.4.4 Code Improvement Implications

The most important result of these improvements is that inserting and deleting a module page only takes a few seconds. Deleting a page is as simple as going to the edit menu and selecting “cut page.” The main menu should then be updated by deleting the page’s link. Adding a page consists of copying an already existing page (dissector or atlas depending on what you are trying to make), and pasting the new page in the appropriate spot. The programmer should then change the page’s name, delete the old text and pictures, and add the new text and pictures. If the programmer needs to create a link to another page, an old link should be copied and pasted on the new page. The name of the
new link should then be changed to the name of the page that it jumps to (remember that the word “link” must precede the destination page name). The main menu should then be updated by adding a link to the new page in the appropriate place.

The other significant advantage of this new code is that adding or deleting a link is trivial. To delete a link, just select it and cut it. To add a link, copy an already existing link, paste it where desired, and change its name as described above.
Chapter 7

The Digital Dissector: Discussion and Conclusions

7.1 Introduction
The first part of this chapter discusses a number of design problems that need to be addressed in future versions of my software. I have divided this part into three sections which address computer, hypermedia, and educational issues respectively. Many of the problems I mention do not have any tried and true solutions, but are topics of ongoing research. The second part of the chapter offers possible directions for future development.

7.2 Computer Related Design Issues
A number of subtle features are lost when educational multimedia is used to replace textbooks. For instance, a textbook is small and can be carried around, while a computer cannot. This means that unlike books, software can only be used in certain places. Furthermore, because the software might keep quiz information and student notes on a public machine, privacy and security need to be addressed. Finally, a textbook is paper so important sentences can be highlighted and notes can be made in the margins. On the other hand, it is difficult to add such features to most computer programs.

7.2.1 Portability
Books can be read anywhere at anytime. They can be used in laboratories without fear of electrical shorts, and small amounts of damage (rips, water, etc.) will not affect their readability. None of these things hold true for software, however, and it is rather difficult to provide solutions to these problems. Ideally, if computers were available everywhere, students could access personalized data over networks or keep it on floppy disks. This is almost never true, though, which makes it difficult to review information outside of the
computer lab. One simple solution is to provide a mechanism for printouts. Alternatively the developer can write programs that augment textbooks rather than replace them.

If the Digital Dissector is ever completed and used in lab, this may turn out to be a significant problem. As described in Chapters 5, I tried to compensate for this by providing a way to print out module pages. However, due to the nuances of Supercard objects, the printouts do not accurately represent what the user sees on screen. As a result, I removed the print feature from the Knee module and this problem is still unsolved. The temporary work around that I have been using is printing screen dumps from other programs, and I am unsure if there is a better solution. Fortunately, students still have their dissection manuals and anatomy atlases.

7.2.2 Privacy and Security

In educational programs, it is often desirable to include features that gather various forms of student data. Privacy and security become important if students do not want their scores or notes to be perused by others. By encrypting data files with a password or limiting student access privileges, designers and administrators can provide a first defense against snoopers. Alternatively, test results can be displayed on the monitor and then discarded. In addition, if usage tracking mechanisms are provided, it may be a good idea to display a notice of this fact so that students know that their progress is being monitored. Unfortunately, it does not seem like privacy issues have been addressed in hypermedia literature, and there are many sticky issues that will need to be dealt with if educational software becomes popular in the classroom.

Because my modules are in the development and proof on concept stages, I did not feel the need to worry about security issues. However, if development proceeds and what I have written becomes the basis for a complete Digital Dissector, privacy of student log files, whether they be quiz or usage tracking files, should be addressed. At the present
time, I am not sure that the Supercard language is powerful enough to provide a means for encrypting or otherwise protecting files.

7.2.3 Annotation

Students like to annotate their textbooks, and surveys have shown that they would also like to annotate educational multimedia programs. As a result, educational program developers should provide some facility for annotation, whether it be completely text based or not. Ideally, besides being able to highlight and annotate like in a textbook, users should be able to add their own links and nodes. The developer must also decide whether annotations are visible to everyone, or only to the student who wrote them. Past studies have indicated that students prefer world visible annotation, but this may depend on the application.

In my opinion, optimal annotation facilities would allow the user to both type and draw on the screen. I attempted to provide this feature in the Knee module, but was unable to get it to work properly. Hopefully if development on my modules continues after I am gone, this problem will be addressed by someone else.

7.3 Hypermedia Issues

Due to the nature of hypermedia documents, it is not uncommon for users to get lost in “hyperspace.” In a network with many nodes and links, users may accidentally find themselves off the path they intended to follow, with no obvious way to get back. Several things can be implemented to avoid this, including maps, backtracking, and bookmarks. Keeping users informed of where links go is also a useful way to keep them from getting lost.

1. Legget, Schnase, & Kacmar, pp 33-34
7.3.1 Hypermedia Maps

When people get lost, they usually employ one of two methods to figure out how to get back to their intended path: 1) ask somebody, and 2) look at a map. The problem with asking someone for directions is that you must know where you want to go, and inevitably, hyperdocument users do not know exactly where that is. Context sensitive or intelligent help facilities might provide a clue for lost users, though. On the other hand, a map can give you an idea of where you are, what is around you, and how to get there. In some instances, as in the case of my Digital Dissector modules, the structure of a hyper-document is almost completely linear, and then map making is primarily a question of format. In other cases, there are enough links in the network that the hyper-document could not even be mapped in 3-space (it would require n-space, n>3). In the latter case, rather than hacking up a 5-dimensional map, the developer’s best bet is probably to identify a number of main paths through the document and map them, or provide a high enough level map that the n-dimensionality of the module disappears.

As far as format is concerned there are a number of options. One popular technique is to provide a graphical representation of the entire network. Alternatively, nodes can be collected into small groups and a multiple level map can be used. Finally, a text outline can be used. This latter approach is common in many of today’s software products and is often used for help utilities. It is also what I used in my software, but I am not convinced that it is an especially good solution. It may be better to implement the map as a network of anatomical pictures (e.g. A picture of a scalpel for the dissection pages, a picture of the knee bones for the bone introduction page, etc.). Unfortunately, no optimal method of presenting a map has been discovered, and the whole problem of getting lost in hyperspace is an area of ongoing research.²

---

1. Mayes, Kibby, & Anderson, pp 233-236
2. Landow, pp 90-94
7.3.2 Backtracking/Forward tracking

A number of popular hypermedia programs provide the user with a backtracking feature. A list is kept of everywhere the user visits so if she suddenly becomes lost, she can backtrack to a familiar node. If the developer chooses to implement a backtracking feature, it may be advantageous to include a way to “forward track” since it is common for users to want to backtrack and look something up, then return to where they started backtracking. If the backtracking feature is implemented using a pushdown stack (like the one I own), it is not possible for users to recover from a backtrack without remembering where they were. This is because once information is popped off the stack, it cannot be recovered. Instead, a better technique might be to grow a list and use a pointer into the list to keep track of where the user is.

Such a feature is rather easy to implement, but not necessarily useful for highly structured modules like the ones I created. I do not expect my choice of implementation to be a problem because the Digital Dissector modules are designed so that the backtracking feature should only be necessary to return from a digression to an atlas page. Otherwise, the user should use the forward and backward buttons. However, users of my software may find that a forward tracking mechanism would be extremely helpful and then someone will have to change my backtracking implementation. This should not be too hard because most of the details have been hidden in the backtrack button’s script.

7.3.3 Bookmarks

In a hypermedia module, users may find one or two particularly helpful pages that they would like to access on a regular basis.\(^1\) In order to make it efficient for users to do this, the developer can allow them to create their own links or provide for a bookmark feature. A bookmark feature for a multimedia program functions just like you would expect; the user “places” a marker somewhere in the module, and can then jump to the marker on

---

1. Legget, Schnase, Kacmar, pp 33-34
demand. This feature can also be used to take users back to a familiar location if they get lost.

Had I thought that my module would have needed something like this, I would have implemented it. However, due to the purpose of my software and the module layout, I did not think that users would want to frequent any particular page. If I am wrong, a bookmark feature can be implemented or the users can be provided with a way to create their own links. Implementation of bookmarks would basically consist of keeping a few extra variables that can contain any page name. Methods can then be provided to set the variables and to jump to their pages.

7.3.4 Link Names

If the best offense is a good defense, the best way to help users get back to where they want to go is to keep them from getting lost. This requires that users know where different links go. Unfortunately, one problem that both the dissection manual and my software share is that it is unclear what information is contained in referenced atlas pages.\(^1\) The dissection manual references atlas sections by their number (e.g. 8.75), but section numbers do not give any indication of actual node content. Since I copied the dissector text, my software has the same problem. Instead, it would be helpful if section numbers were replaced by section titles (e.g. "Bones of the Knee"), or there was some way to display this information (e.g. holding down the option key and clicking the link). The former is better from a user standpoint, but if this problem really is not an issue, the latter would be easier to implement.

---

1. Landow, p 96-98
7.4 Educational Issues

7.4.1 An Aid for Students

Of course, the primary use of an educational multimedia program is as an aid for students. As I have discussed in previous chapters, the specific lesson determines how the software should be structured. In instances where student directed learning is optimal, the module should be very unstructured as long as precautions have been taken against cognitive overhead. The student can then explore areas she finds interesting and will be more likely to benefit from the software. On the other hand, if the lesson needs to be highly structured as in my case, special attention should be paid to keeping students from getting lost or sidetracked. To be of use as an educational tool it is also important that the software does not detract from the learning process. This is only possible if the program’s interface is simple and easy to use, and minimizes the danger of getting lost, information overload, boredom, etc.

I think that the structure and format of my module is fairly good, but only testing in the lab will confirm this. If I was going to be around for another year, I would be able to oversee testing of the software myself. However, this will have to be handled by someone else. Probably the best way to get an idea of what needs to be fixed is to watch students using the software in lab and ask them what they like and dislike. It is important to watch the students because an observer may notice that they have bad habits, use the software in an inefficient manner, or do not take advantage of features they would find helpful (and users tend not to report these kinds of difficulties). The software can then be redesigned in order to minimize any problems.

Other aspects of education that need to be applied to multimedia are those that help the student learn and target weaknesses. Providing a way for students to test their knowledge can help them determine which of their conceptual models need refinement. An alternative
solution is to add intelligence to the software so that it can determine which paths through the module are best for the student given past actions, test results, etc. This could be implemented in my software by providing a few parallel paths for students to follow, differing only in complexity or depth of the material presented. Short quizzes could be used to establish the knowledge of the user and the software could then direct the user down the path best suited to her needs. However, I feel that this would be difficult to do given the goals of my software. Instead, it would be better to make illustrative video clips or animations available to the user.

7.4.2 An Aid for Teachers
While it is primarily intended as a learning tool for students, the software can (and should) also be developed as a teaching tool for faculty. Where a learning tool facilitates absorption and understanding on the part of the student, a teaching tool provides a mechanism for teachers to evaluate the progress of their students and adapt their lesson plans accordingly. This can be done through a variety of quizzing and usage tracking techniques, and the fact that everything is on computer means that testing mechanisms can be much more dynamic than usual. However, my software only provides one mode of testing and one mode of usage tracking. Once the software is used in lab, though, feedback from students and teachers can help decide what types of quizzes are best and what events need to be tracked. At that time, improved features can be implemented and the old modules can be retrofitted.

7.5 Advice for Other Developers
Before concluding I would like to mention a few things about design. Software design is one of those subjects that has lots of literature, yet it seems that 99% of everything written is difficult to use and non-intuitive. I personally think that most secrets to software design are just common sense, but that it is easy concentrate on producing something
functional rather than something functional and easy to use. Here I only want to discuss
one general design rule, and a short list of interface guidelines.

7.5.1 KISS

Probably one of the most useful development rule is “keep it simple, stupid” which,
when exercised, gets rid of many “features” that are likely to be cause for confusion.
Through proper application it guarantees that commands will be easy to execute (i.e. NOT
type-control-shift-x 4) and easy to find (i.e. NOT buried three menus deep). It also insures that
status messages are clear and informative (i.e. NOT “system error # 234) and user
interfaces are simple and intuitive.

As far as my software was concerned, I used the KISS principle as a check for the
design decisions I made. Although a few of the things I ended up doing were probably not
in the user’s best interest, for the most part I managed to keep the interface simple. The
number of controls I ended up using was rather small, and all of the features I
implemented were straight forward. Usually the simpler a program is, the easier it will be
to use and the more user friendly it will be.

7.5.2 Principles of Good Design

Below is a self explanatory list of “principles of good design” from a user interface
textbook¹ along with some examples of how they relate to my software. The idea is that if
your software does its best to meet theses guidelines, odds are it will be very easy to use.

- Feedback: the watch icon is displayed while the system is active (which is not often).
- Status: page titles tell the user where she is, the cursor changes when something is
clickable, forward and backward page controls are missing from atlas pages, links
change color when they have been used.
- Minimal work: all controls involve buttons or clickable objects, links are easy to find,
controls are easily accessible, the module is primarily linear requiring only short
digressions if any.
- Default options: none necessary.
- Help: help pages accessible from every page, balloon help, instructions on screen in

¹. Sutcliffe, pp 101-102
many places, annotation mechanism for student clarifications, dissector page locator.
- Undo: backtracking mechanism, button to resent link colors in the help pages.
- Consistency: same buttons used throughout the module, page layout is the same from
  page to page, all page links are the same color (red), all display links are the same
  color (blue), all animation icons are the same, anatomy menus (when used) match
  those used in A.D.A.M.

7.6 Future Extensions

Although my software is just two modules, it can easily be expanded in a number of
ways. If development continues along the path that I have been taking, the end result will
be a multimedia dissector/atlas combination. Towards this end, I have provided a
functional framework for future modules so that everything is in place except for text,
pictures and animations; in addition to my Larynx and Knee modules, I have also written a
short template module which can be used as the basis for new chapters of the Digital
Dissector. It should be possible to construct a new module simply by copying the template
and filling in missing information. The template contains menu, introduction, dissector,
atlas, quiz, help, and programmers pages. By copying these pages, filling them with new
information, updating page names, and so on, module creation should be easy and
straightforward. However, future programmers should note that image editing takes up a
significant fraction of module development time. Instructions for using the template are in
appendix C.

However, there is no reason to stop there. The most obvious direction to go is toward
the development of a complete multimedia “textbook.” Modules can be developed one at a
time until the entire book is complete. Even if implementation details change from module
to module, a high level menu can be used to enter into individual modules. Links can be
added between modules to create a unified whole. Depending on the number of
interconnects, the result can be anything from a “stream of consciousness” hyperbook to a
A second possible addition would be a “virtual dissector.” This is something that I originally hoped to do, but it is quite an undertaking. By virtual dissector, I mean a graphical body on screen which one can dissect using graphical tools. The A.D.A.M. software makes a crude attempt at this, but is really nothing more than an anatomy atlas. What I have in mind is something that simulates real tools and behaves just like a real body. For instance, skin and organs should be able to be cut and then folded back. You should be able to roll the body or operate from different angles. You should be able to use screws, sutures, and other medical devices. The software should allow the student to have as real of an experience as possible. By doing so, students will be able to go through numerous practice sessions, a tremendous amount of money can be saved, and first time experiences on real bodies will be much easier. Such a program will also allow teachers to demonstrate surgical procedures or dissection techniques in class.

On the other hand, if it turns out that using multimedia to replace or augment dissection manuals is not really that great of an idea, it can still be used to teach anatomy and surgical techniques. A.D.A.M. is a good first step towards teaching anatomy, but it needs some sort of intelligent guide to direct the student. Just learning the names of anatomical parts is not that useful, either. What would be better is something along the lines of the common injuries section of my Knee module which graphically describes surgical techniques and has helpful animations.

7.7 Conclusion

In this thesis I have described the development and implementation of two educational multimedia modules designed to help medical students learn anatomy and dissection. The
first of these was really only a proof of concept, while the second is functionally complete
and can be used as a model for future modules. My software has been designed around
three main goals, namely 1) it should help students finish their dissection labs, 2) it should
be easy to use, and 3) it should be easy to expand and maintain. It is my hope that
development efforts will continue, and that eventually the Digital Dissector modules will
actually be used in lab. At that time it will become apparent if the software is as easy to
use as I hope it is, and whether it is useful to students. During my research both on and off-
line, I did not find any indication that software similar to what I have developed exists or is
being developed.

While describing the software that I have written, I have also mentioned a large
number of design issues relevant to educational multimedia. These issues range from
everything to how a module should be structured, to page layout, to link features, and so
on. Ideally, the reader has come away with a number of ideas as to how an educational
multimedia module should be engineered and what some of the relevant factors are. As a
result, I hope I have given something to both medical students and software developers.
Appendix A

Pictures from the Digital Dissector Modules

This appendix contains seventeen pictures from the Larynx and Knee modules. These pictures were created from screen dumps and were scaled and edited using a variety of graphics programs. The low resolution of some pictures is a result of scaling. The images are as follows:

A.1 The A.D.A.M. Interface
A.2 The Larynx Module Main Menu
A.3 The Larynx Module Introduction Page
A.4 A Larynx Module Dissector Page
A.5 A Larynx Module Atlas Page
A.6 The Larynx Module Help Page One
A.7 A Knee Module Introduction Page With Annotation Field On
A.8 A Knee Module Dissector Page
A.9 A Knee Module Common Procedures Page
A.10 A Knee Module Atlas Page
A.11 The Knee Module Quiz Page
A.12 The Knee Module Programmers Page
A.13 The PICT Viewer
A.14 An Animation, Courtesy of Dr. Ratiu
A.15 An Animated Scan, Courtesy of Dr. Ratiu
A.16 The Director Movie Viewer
A.17 The Template Module Menu

1. Atlas Page 8.72 has been visited even though the light pink does not show up on this picture. The Cricothyroid box has also been clicked.
Main Menu:

1. Dissection of Larynx
2. Dissection continued

Relevant Atlas Pages:
- Skeleton of the larynx, posterior view
- Muscles of the pharynx, larynx, and esophagus
- Skeleton of the Larynx
- Muscles and nerves of the larynx, cricothyroid joint
- Thyroid cartilage and cricothyroid muscle
- Larynx
- Interior of the larynx
- Larynx, superior view
- Skeleton of the Larynx
- Epiglottis and hyoepiglottic ligament
- Actions of the laryngeal muscles
- Transverse MRI of the larynx through the thyroid cartilage
- Transverse MRI of the larynx through the cricoid cartilage
- Compartments of the larynx, coronal section
- Coronal MRI through the oropharynx, larynx, and trachea
- Distribution of the vagus nerve
The skeleton of the larynx (Atlas) is responsible for maintaining the shape of this organ. It consists of a series of articulating cartilages which are united by membranes. The epiglottic cartilage (Gk., krikos, ring) is shaped like a signet ring; its large plate or lamina is positioned posteriorly, its arch anteriorly.

The inferior horns or cornua of the thyroid cartilage articulate with the cricoid cartilage at special facets. At these facets, the cornua can be tilted forward or backward in a visor-like manner.

On the upper border of the lamina of the cricoid are the articular facets for the paired arytenoid cartilages. These small pyramidal cartilages are capable of:

1. Tilting forward and backward.
2. Sliding toward or away from another.
3. Rotary motion.

The posterior ends of the vocal ligaments are attached at the vocal processes of the arytenoid cartilages (Atlas). The anterior ends of the vocal ligaments converge at the angle formed by the laminae of the thyroid cartilage.

The epiglottic cartilage lies posterior to the tongue and (Atlas). The stalk of this cartilage is attached in the angle between the thyroid laminae, just superior to the vocal ligaments (Atlas).

Hold down the mouse button over parts of the larynx to identify them.
Define the \textbf{recurrent laryngeal nerve} [Atlas 8.72]. Just posterior to the joint runs the recurrent laryngeal nerve. Cut through the ligamentous bands which hold the joint together. Disarticulate this synovial joint.

The next objective is to reflect a portion of the thyroid lamina in order to expose the remaining laryngeal muscles (atlas 8.72). Proceed as follows: Saw or cut the lamina of the thyroid cartilage about 8 mm to the left of the midline. Reflect the thyroid lamina which is attached to the cricoid by the cricothyroid muscle.

Now, the following laryngeal muscles can be identified and studied (atlas 8.72)

1) \textbf{stretching from the surface of the cricoid to the lower border and inferior horn of the thyroid cartilage.}

2) \textbf{passing from the upper border of the cricoid and the cricothyroid ligament to the muscular process of the arytenoid cartilage.}

3) \textbf{positioned superior to the lateral cricoarytenoid. It passes from the thyroid cartilage anteriorly to the arytenoid cartilage posteriorly. Its upper and most medial fibers are the vocalis.}

4) \textbf{applied lateral and inferior to the vocal ligament (atlas 8.72).}

5) \textbf{from thyroid cartilage to epiglottis (atlas 8.72).}
Superior to the vocal folds (vocal cords), the larynx is sectioned near the median plane and the interior of its left side is seen. Inferior to this level, the right side of the larynx is dissected.

Observations:
1. The hypoepiglottic ligament and the thyrohyoid membrane, both attached to the superior part of the body of the hyoid bone.
2. The tissue and the collection of glands (not labeled) filling the triangular space.

Epiglottic cartilage
- Aryepiglottic fold
- Cuneiform tubercle
- Corniculate cartilage
- Triangular pit
- Vocal process
- Muscular process
- Lamina of cricoid cartilage
- Facet for cricoid cartilage

Hypoepiglottic ligament
- Hyoid bone
- Thyrohyoid
- Vestibular fold
- Ventricles of larynx
- Vocalis
- Vocal fold
- Cricothyroid ligament
- Median cricothyroid ligament
- Articulation of cricoid

Previous Page  Menu  Print  Help!
The larynx module is currently under development so things may be a little buggy. Turn on Balloon Help for more info. The basic features should work on all screens. Right now they include:

1) Clicking on a term in a light blue box will toggle a circle surrounding the body part referred to by that term. Click on this box to toggle a circle.

2) Clicking on a number or word in a pink box will take you to another page (usually a relevant atlas page). After you have been to a page once, links that take you back to it become light pink so that you can tell where you’ve been. Click the Clear Links button to reset the colors.

3) Clicking on a QuickTime icon will show you a movie about the text or object which it is next to. Click on this icon to see a sample movie.

4) Clicking on a Picture icon will show you a picture of the text or object which it is next to. Click on this icon to see a sample picture.

5) Clicking on the Menu button at the bottom of the screen will take you to the main menu of the larynx module. Clicking on this Menu button will take you to the Main menu.

6) Clicking on the arrow buttons will take you forward/backward one page in the current sequence of pages. Some pages do not have arrow buttons because they are not part of a logical sequence. Clicking on these buttons will not do anything.

7) Clicking on the Previous Page button will take you back to the last page you were on (except if it was the Help page, in which case it takes you to the page before). After warping to an Atlas page (clicking on a pink box) you can use this button to return to the page you warped from. Clicking on this Previous Page button will not do anything.

8) Clicking on the Help button will take you to this page.
There are four bones in the general vicinity of the knee: the femur, tibia, fibula, and patella. The articulating surfaces of these bones are protected by cartilage.

**Annotation field**

Wow! Take a look at the bones animation! Notice that all the knees featured on this page are right knees. -CH

The next page has information about the menisci which also protect the articulating surfaces of the femur and tibia. -JU
On the anterior aspect of the knee joint, identify the expansions of the vasti muscles and the ligamentum patellae. Palpate the patella. Verify the existence of the prepatellar bursa just anterior to the patella. Detach the quadriceps tendon from the patella. Be careful not to damage the underlying synovial capsule of the knee joint (Atlas, synovial capsule shown in blue). Make a transverse cut immediately superior to the patella through the synovial capsule into the joint cavity. With a blunt instrument, carefully explore the extent of the capsule. Note the superior recess of the joint cavity. This sac-like recess is the suprapatellar or quadriceps bursa.

Just superior to the quadriceps bursa, make a wide horseshoe-shaped section through the quadriceps from epicondyle to epicondyle. Identify the quadriceps bursa. Note a layer of fat (fat pad) between bursa and femur.
Click here to see a lateral meniscus through a real arthroscope!

Cross section through knee

Arthroscope inserted in the knee

Attachments of tear are divided and cartilage fragment is removed.

Menu  Backtrack  →  Toggle annotation  Help!
Quiz Instructions:

Type "quit" at any time to stop the quiz.

Clicking "Cancel" or clicking "OK" without an answer will display a question.

Quiz on the Bones of the Knee
<table>
<thead>
<tr>
<th>make quiz files</th>
<th>View Student usage log</th>
</tr>
</thead>
<tbody>
<tr>
<td>close the quiz file</td>
<td>View all page names</td>
</tr>
<tr>
<td></td>
<td>Show a page script</td>
</tr>
<tr>
<td></td>
<td>Change Page/Link Scripts</td>
</tr>
</tbody>
</table>

```
-- moving into the link, when Balloon help is on, you'll get a message. This
-- is sneaky because
-- the current page names are not meaningful to the average user (the names
aren't the same as
-- the page titles at the top of the screen).

on mouseEnter
    set lockCursor to "true"
    set cursor to arrow
    if the shiftKey is "down" then
        FullBalloons "ShowDirect","Jump to page "&(word 2 of the short name of
me)
    end if
end mouseEnter

on mouseLeave
    set lockCursor to "false"
    set cursor to hand
end mouseLeave
```
There are four bones in the general vicinity of the knee: the femur, tibia, fibula, and patella. The articulating surfaces of these bones are protected by cartilage.
The next few pages contain the instructions for...
The anterior attachment is divided
Appendix B

Code Listings, Larynx Module

B.1 Larynx Project

--L* are the variables which keep track of where the user has been.
--Initialize them to false

on openProject
    global prevPage, L8.69, L8.70, L8.71, L8.72, L8.73, L8.74, L8.75, L8.76,
    L8.77, L8.78, L8.79, L8.81.1, L8.81.2, L8.82, L8.83, L9.15.2
    global L.General_Remarks, L.Laryngeal_Muscles, L.Laryngeal_Muscles.2,
    L.Interior_of_the_Larynx, L.Interior_of_the_Larynx.2,
    L.Interior_of_the_Larynx.3, L.Laryngeal_Muscles.3,
    start_time, name_string, l.gr.t, l.lm.t, l.lm2.t, l.lm3.t, l.il.t,
    l.il2.t, l.il3.t, l.h.t, l.a.t
    put Larynx_Menu into prevPage
    put "false" into L8.69
    put "false" into L8.70
    put "false" into L8.71
    put "false" into L8.72
    put "false" into L8.73
    put "false" into L8.74
    put "false" into L8.75
    put "false" into L8.76
    put "false" into L8.77
    put "false" into L8.78
    put "false" into L8.79
    put "false" into L8.81.1
    put "false" into L8.81.2
    put "false" into L8.82
    put "false" into L8.83
    put "false" into L9.15.2
    put "false" into L.Interior_of_the_Larynx
    put "false" into L.Interior_of_the_Larynx.2
    put "false" into L.Interior_of_the_Larynx.3
    put 0 into l.gr.t
    put 0 into l.lm.t
    put 0 into l.lm2.t
    put 0 into l.lm3.t
    put 0 into l.il.t
    put 0 into l.il2.t
    put 0 into l.il3.t
    put 0 into l.h.t
put 0 into l.a.t
put seconds() into start_time
ask “What is your name?”
put it into name_string
end openProject

-- l.gr.t => larynx.General_remarks.time

on closeProject
    global name_string, start_time, l.gr.t, l.lm.t, l.lm2.t, l.lm3.t,
    l.il.t, l.il2.t, l.il3.t, l.h.t, l.a.t
    -- Update log file
    put name_string&"":"&(seconds()-start_time) into temp
    put " GR:"&l.gr.t&" LM:"&l.lm.t&" LM2:"&l.lm2.t&" LM3:" &l.lm3.t&" IL:"&l.il.t after temp
    put " IL2:"&l.il2.t&" IL3:"&l.il3.t&" ATL:"&l.a.t&" HELP:" &l.h.t after temp
    put "Larynx.log" into fName
    savefile fName temp
end closeProject

on savefile fName,temp
    global textChange,adamShell,adamAuthor,shellPath
    put shellPath&"ADAM Projects:ZLARYNX picts:"&fName into fName
    put "" into old_stuff
    -- Put temp into the file named fName
    open file fName
    put 1 into it
    repeat until it is empty
    read from file fName for 12000
    put it after old_stuff
    end repeat
    write old_stuff&cr&temp to file fName
    close file fName
end savefile

B.2 First Dissector Page in the Larynx Module

on openCard
    global moduleView,curInk,adamShell,saveFlag, GR_1_H, GR_2_H, GR_3_H
    global L8.69, L8.70, L8.71, L8.72, L8.73, L8.74, L8.75, L8.76, L8.77,
    L8.78, L8.79, L8.81.1, L8.81.2, L8.82, L8.83, L9.15.2
    global L.General_Remarks, timer.temp
    -- We’ve been to this page
    put "true" into L.General_Remarks
-- Start the timer to measure how long the user's on this page
put seconds() into timer.temp

-- We need to hide the circles around the anatomy parts until the user
-- clicks the blue rectangles.
-- I'm not sure if the saveFlag, topWindow(), and lockUp are necessary.
put "true" into saveFlag
setWindow topWindow()
send lockUp to proj adamShell
lock screen
put "true" into GR_1_H
put "true" into GR_2_H
put "true" into GR_3_H
hide cd grc GR_1
hide cd grc GR_2
hide cd grc GR_3

-- If the user has been to a page that has a link on this page
-- (a red box), then lighten the color
if L8.69 then set fillFore of grc 13 of this cd to 7
else set fillFore of grc 13 of this cd to 19

if L8.71 then set fillFore of grc 8 of this cd to 7
else set fillFore of grc 8 of this cd to 19

if L8.77 then set fillFore of grc 9 of this cd to 7
else set fillFore of grc 9 of this cd to 19

if L8.77 then set fillFore of grc 10 of this cd to 7
else set fillFore of grc 10 of this cd to 19

if L8.73 then set fillFore of grc 11 of this cd to 7
else set fillFore of grc 11 of this cd to 19

if L8.78 then set fillFore of grc 12 of this cd to 7
else set fillFore of grc 12 of this cd to 19

if L8.81.1 then set fillFore of grc 15 of this cd to 7
else set fillFore of grc 15 of this cd to 19

if L8.81.2 then set fillFore of grc 16 of this cd to 7
else set fillFore of grc 16 of this cd to 19
unlock screen
end openCard

on closeCard
-- Update how long we've spent on this page
global timer.temp, l.gr.t
put seconds() - timer.temp into temp
put l.gr.t + temp into l.gr.t
end closeCard

B.3 Larynx Menu

on openCard
  global L8.69, L8.70, L8.71, L8.72, L8.73, L8.74, L8.75, L8.76, L8.77,
  L8.78, L8.79, L8.81.1, L8.81.2, L8.82, L8.83, L9.15.2

  global L.General_Remarks, L.Laryngeal_Muscles, L.Laryngeal_Muscles.2,
  L.Interior_of_the_Larynx, L.Interior_of_the_Larynx.2,
  L.Interior_of_the_Larynx.3, L.Laryngeal_Muscles.3

  lock screen

  if L.General_Remarks then set fillFore of grc 1 of this cd to 7
  else set fillFore of grc 1 of this cd to 19

  if L.Laryngeal_Muscles then set fillFore of grc 2 of this cd to 7
  else set fillFore of grc 2 of this cd to 19

  if L.Laryngeal_Muscles.2 then set fillFore of grc 20 of this cd to 7
  else set fillFore of grc 20 of this cd to 19

  if L.Laryngeal_Muscles.3 then set fillFore of grc 23 of this cd to 7
  else set fillFore of grc 23 of this cd to 19

  if L.Interior_of_the_Larynx then set fillFore of grc 3 of this cd to 7
  else set fillFore of grc 3 of this cd to 19

  if L.Interior_of_the_Larynx.2 then set fillFore of grc 21 of this cd to 7
  else set fillFore of grc 21 of this cd to 19

  if L.Interior_of_the_Larynx.3 then set fillFore of grc 22 of this cd to 7
  else set fillFore of grc 22 of this cd to 19

  if L8.69 then set fillFore of grc 4 of this cd to 7
  else set fillFore of grc 4 of this cd to 19

  if L8.70 then set fillFore of grc 5 of this cd to 7
  else set fillFore of grc 5 of this cd to 19

  if L8.71 then set fillFore of grc 6 of this cd to 7
  else set fillFore of grc 6 of this cd to 19

  if L8.72 then set fillFore of grc 7 of this cd to 7
  else set fillFore of grc 7 of this cd to 19

  if L8.73 then set fillFore of grc 8 of this cd to 7
  else set fillFore of grc 8 of this cd to 19
if L8.74 then set fillFore of grc 9 of this cd to 7
else set fillFore of grc 9 of this cd to 19

if L8.75 then set fillFore of grc 10 of this cd to 7
else set fillFore of grc 10 of this cd to 19

if L8.76 then set fillFore of grc 11 of this cd to 7
else set fillFore of grc 11 of this cd to 19

if L8.77 then set fillFore of grc 12 of this cd to 7
else set fillFore of grc 12 of this cd to 19

if L8.78 then set fillFore of grc 13 of this cd to 7
else set fillFore of grc 13 of this cd to 19

if L8.79 then set fillFore of grc 14 of this cd to 7
else set fillFore of grc 14 of this cd to 19

if L8.81.1 then set fillFore of grc 15 of this cd to 7
else set fillFore of grc 15 of this cd to 19

if L8.81.2 then set fillFore of grc 16 of this cd to 7
else set fillFore of grc 16 of this cd to 19

if L8.82 then set fillFore of grc 17 of this cd to 7
else set fillFore of grc 17 of this cd to 19

if L8.83 then set fillFore of grc 18 of this cd to 7
else set fillFore of grc 18 of this cd to 19

if L9.15.2 then set fillFore of grc 19 of this cd to 7
else set fillFore of grc 19 of this cd to 19

unlock screen
end openCard

B.4 Larynx Atlas Page
-- Start timer when opened, update atlas time when closed.
on openCard
  global timer.temp
  set lockCursor to "false"
  set cursor to hand
  put seconds() into timer.temp
end openCard

on closeCard
  global timer.temp, l.a.t
  put seconds()-timer.temp into temp
  put l.a.t+temp into l.a.t
end closeCard
B.5 ADAM Style Pop-up Script for a Graphic

Thyroid Cartilage, 2, Histology, Radiology
PICT, Macintosh HD: A.D.A.M. Folder: Histology: HS8.08, HS8.08, ?, ?, 1/17/94
Superior pharyngeal constrictor muscle, 1, Histology
Hyoid bone, 1, Histology
Medial cricothyroid ligament, 1, Histology
Thyrohyoid membrane, 1, Histology
Greater horn of the hyoid bone, 2, Histology, Radiology
Superior horn of the thyroid cartilage, 2, Histology, Radiology
PICT, Macintosh HD: A.D.A.M. Folder: Histology: HS8.08, HS8.08, ?, ?, 1/17/94
Lesser horn of hyoid bone, 1, Histology

END

-- The format for the beginning of this script is as follows:
--
-- [Name of the region], [number of options under More Info],
-- [Name of option 1], [Name of option 2], ...
-- [file info for option 1]
-- [file info for option 2]
-- ...
-- [Name of next region], [...]
-- ...
-- END
--
-- Note that the name of the region should be *EXACTLY* the same as the
-- text in the mouseDown handler

on mouseEnter
    set lockCursor to "true"
    set cursor to arrow
end mouseEnter

on mouseLeave
    set lockCursor to "false"
    set cursor to hand
end mouseLeave

on mouseDown
    if the optionKey is "down" then
        answer item 1 of the clickLoc - the left of me
        answer item 2 of the clickLoc - the top of me
    end if
-- This defines the regions where clicking the mouse gets a menu. I put
-- the name of the region into a variable and call the display function
-- to handle the rest.
put "" into areaName
put item 1 of the clickLoc - the left of me into x
put item 2 of the clickLoc - the top of me into y

if (x > 168) and (x < 266) then
if (y < 65) then put "Superior pharyngeal constrictor muscle" into areaName
if (y > 83) and (y < 104) then put "Hyoid bone" into areaName
end if

if (x > 165) and (x < 174) then
if (y > 61) and (y < 86) then put "Greater horn of the hyoid bone" into areaName
if (y > 112) and (y < 132) then put "Superior horn of the thyroid cartilage" into areaName
end if

if (y < 92) then
if (y > 71) then
if (x > 175) and (x < 196) then put "Lesser horn of hyoid bone" into areaName
if (x > 242) and (x < 261) then put "Lesser horn of hyoid bone" into areaName
end if
if (y > 64) and (x > 199) and (x < 240) then put "Epiglottis" into areaName
if (y > 59) and (x > 266) and (x < 275) then put "Greater horn of the hyoid bone" into areaName
end if

if (x > 208) and (x < 230) and (y > 211) and (y < 229) then put "Arch of cricoid cartilage" into areaName

if (y > 190) and (y < 211) then
if (x > 231) and (x < 252) then put "Cricothyroid Muscle" into areaName
if (y > 104) then
if (x > 233) and (x < 266) and (y > 126) and (y < 129) then put "Cricothyroid Muscle" into areaName
if (x > 204) and (x < 234) and (y > 131) then put "Medial cricothyroid ligament" into areaName
end if

if (y < 188) then
if (x > 173) and (x < 207) and (y > 129) then put "Thyroid Cartilage" into areaName
if (x > 206) and (x < 234) and (y > 141) then put "Thyroid Cartilage" into areaName
if (x > 233) and (x < 266) and (y > 130) then put "Thyroid Cartilage" into areaName
end if

if (x > 265) and (x < 269) and (y > 115) and (y < 129) then put "Superior horn of the thyroid cartilage" into areaName
if (x > 170) and (x < 190) and (y > 102) and (y < 122) then put "Thyrohyoid membrane" into areaName
if (x>189) and (x<205) and (y>106) and (y<126) then put "Medial cricothyroid ligament" into areaName
if (x>211) and (x<231) and (y>189) and (y<207) then put "Median cricotracheal ligament" into areaName

if areaName is not empty then displayInfo areaName
end mouseDown

on displayInfo areaName
  -- Displays the proper menu depending on the info at the beginning of this script
  -- First, we want to find the line above that starts with the right areaName,
  -- that way we can get the options that should go in the menu. If no such line
  -- exists, then just print the name of the area.
  put 1 into index
  put "false" into quitLoop
  put item 1 of line index of the script of me into currentItem
  repeat until quitLoop
    if currentItem = areaName then
      put "true" into quitLoop
    else if currentItem = "END" then
      put "true" into quitLoop
    else
      put index + 1 into index
      put item 1 of line index of the script of me into currentItem
    end if
  end repeat

  put areaName into choices
  -- No information about this area in the listing at the beginning of the script,
  -- therefore just print the area name..
  put "<B"&cr after last item of line 1 of choices
  if currentItem = "END" then
    get FullHPop(choices,the mouseloc,1, geneva ,9)
    exit displayInfo
  else
    -- Found info about the area, therefore get the options from the appropriate
    -- line in the script and put them into the menu.
    put item 2 of line index of the script of me into numPicts
    put "-"&crk"More Info," after choices
    put 0 into pictIndex
    repeat until pictIndex = numPicts
      put item 3+pictIndex of line index of the script of me after choices
      put "," after choices
      put pictIndex + 1 into pictIndex
    end repeat
    -- Display the menu
    put FullHPop(choices,the mouseloc,1, geneva ,9) into hit

    if item 1 of hit is not "More Info" then bye
    if item 2 of hit is empty then bye
    put 0 into pictIndex
    put "false" into quit
    -- Go through the list of options until you find the selected one.
    -- This gets you the offset of that option, which you
    -- can then use to find the proper file to display. The file info is offset the
    -- same amount from the current line.
    repeat until quit
    end displayInfo

if item 2 of hit = item 3+pictIndex of line index of the script of me then put "true" into quit
put pictIndex + 1 into pictIndex
if pictIndex = numPicts then put "true" into quit
end repeat
-- For the time being, we're only using the menu to display pict. This can
-- change in the future simply by writing handlers
-- that look at what the first item in the line is and responding appropriately.
-- Maybe something to show movies, etc.
if item 1 of line index+pictIndex of the script of me = PICT then displayPICT
(line index+pictIndex of the script of me),(the long id of me)
end if
end displayInfo

--Variable list:
--areaName = the name of the area that the user has clicked on. We're trying to
--find its info at the begining of the script

--index = current line in the area info at the beginning of the script. Once
--you've found the right line for the area the user has clicked on, you can
--offset from it to find the appropriate PICTS, depending on what menu option
--they choose.

--currentItem = the name of the area we're currently looking at

--choices = the variable that's used to make the menu. Shoving stuff in here gets
--it displayed

--numPicts = the number of options the user has for this given area (item 2 of
--the appropriate line)

--pictIndex = current option that we're looking at. Used to find the one the user
--has selected from the menu, and therefore the offset down from line index to
--the appropriate file info.

B.6 Forward Button ("->")

on mouseUp
    global moduleView, curInk, adamShell, saveFlag, prevPage
    -- Declaration of global variables above.
    put "true" into saveFlag
    setWindow topWindow()
    send lockUp to proj adamShell
    -- Update the Previous Page variable
    put General_Remarks into prevPage
    -- Go to the next page
    go next cd of wd 1 of this proj
    unlock screen
end mouseUp

on mouseEnter
    set lockcursor to false
    FullBalloons "ShowDirect","Go forward one page."
end mouseEnter
B.7 Menu Button

on mouseUp
  global moduleView, curInk, adamShell, saveFlag, prevPage
  put "true" into saveFlag
  setWindow topWindow()
  send lockUp to proj adamShell
  -- Update the previous page variable
  put General_Remarks into prevPage
  -- Go to the main menu
  go cd Larynx_Menu of wd 1 of this proj
  unlock screen
end mouseUp

on mouseEnter
  set lockcursor to false
  FullBalloons "ShowDirect","Go to the Main Menu"
end mouseEnter

B.8 Previous Page

on mouseUp
  global moduleView, curInk, adamShell, saveFlag, prevPage
  put "true" into saveFlag
  setWindow topWindow()
  send lockUp to proj adamShell
  -- Go to the page kept in the previous page variable
  go cd prevPage of wd 1 of this proj
  put General_Remarks into prevPage
  unlock screen
end mouseUp

on mouseEnter
  set lockcursor to false
  FullBalloons "ShowDirect","Go to the last page you were at."
end mouseEnter

B.9 Help Button

on mouseUp
  global moduleView, curInk, adamShell, saveFlag, helpPage, H_1_H
  put "true" into saveFlag
  setWindow topWindow()
  send lockUp to proj adamShell
  -- Update the help variable (so we know where to return)
  put General_Remarks into helpPage
  -- Go to the help page (only one right now)
  go cd Help of wd 1 of this proj
  -- Signal that we want to reset the help page features
  put "true" into H_1_H
  unlock screen
end mouseUp
on mouseEnter
    set lockcursor to false
    FullBalloons "ShowDirect","Go to the Help pages."
end mouseEnter

B.10 Link Object (to Atlas Page 8.69)

on mouseUp
    global moduleView, curInk, adamShell, saveFlag, prevPage, L8.69
    put "true" into saveFlag
    setWindow topWindow()
    -- Lock everything so the user can't mess stuff up and
    -- screen modifications happen at the same time
    send lockUp to proj adamShell
    -- We need to keep track of where we are, so we can get
    -- back using the previous page button
    put General_Remarks into prevPage
    -- We also need to keep track of which pages we've been
    -- to so we can change the link color
    put "true" into L8.69
    go cd 8.69 of wd 1 of this proj
    unlock screen
end mouseUp

B.11 Larynx Module “Clear Links” Button (on the help page)

on mouseUp
    global L8.69, L8.70, L8.71, L8.72, L8.73, L8.74, L8.75, L8.76, L8.77,
    L8.78, L8.79, L8.81.1, L8.81.2, L8.82, L8.83, L9.15.2
    global L.General_Remarks, L.Laryngeal_Muscles, L.Laryngeal_Muscles.2,
    L.Interior_of_the_Larynx, L.Interior_of_the_Larynx.2,
    L.Interior_of_the_Larynx.3
    put "false" into L.General_Remarks
    put "false" into L.Laryngeal_Muscles
    put "false" into L.Laryngeal_Muscles.2
    put "false" into L.Laryngeal_Muscles.3
    put "false" into L.Interior_of_the_Larynx
    put "false" into L.Interior_of_the_Larynx.2
    put "false" into L.Interior_of_the_Larynx.3
    put "false" into L8.69
    put "false" into L8.70
    put "false" into L8.71
    put "false" into L8.72
    put "false" into L8.73
    put "false" into L8.74
    put "false" into L8.75
    put "false" into L8.76
    put "false" into L8.77
put "false" into L8.78
put "false" into L8.79
put "false" into L8.81.1
put "false" into L8.81.2
put "false" into L8.82
put "false" into L8.83
put "false" into L9.15.2
lock screen
set fillFore of grc 5 of this cd to 19
unlock screen
end mouseUp

on mouseEnter
    set lockcursor to false
    FullBalloons "ShowDirect", "Click on me to reset the color of all the
    link boxes."
end mouseEnter

B.12 Larynx Anatomical “Highlight” Links (the blue boxes)

--1,Show,Plain,Very Fast- This must be the first line of this script_
--GR= General Remarks 1= graphic 1 H=Hidden
on mouseUp
    global moduleView, curInk, adamShell, saveFlag, GR_1_H
    put "true" into saveFlag
    setWindow topWindow()
    send lockUp to proj adamShell
    -- This just toggles whether or not we show/hide the circle
    if GR_1_H then
        put "false" into GR_1_H
        show cd grc GR_1
    else
        put "true" into GR_1_H
        hide cd grc GR_1
    end if
    set cursor to arrow
    unlock screen with Plain Very Fast
end mouseUp

on mouseEnter
    set lockCursor to "true"
    set cursor to arrow
end mouseEnter

on mouseLeave
    set lockCursor to "false"
    set cursor to hand
end mouseLeave
B.13 Anatomic “Highlight” Circle (the object the blue boxes toggle)

--1,Hideable,Plain,Very Fast- Must be in first line of a hideable script_
-- User only clicks on the circle if its in view, therefore we hide it
-- and let the blue rect. know that the circle’s hidden
on mouseUp
  global GR_1_H
  lock screen
  hide me
  put "true" into GR_1_H
  unlock screen with Plain Very Fast
end mouseUp
Appendix C

The Knee

C.1 The Knee Project Script

--link_object is a variable used to keep track of what pages have been
--visited. It has a number of slots equal to the highest id number of any
--page in the project.
--
--visitLink is used when a link has been or will be visited. It indexes
--into link_object using the id number of the page we’re linking to and
--sets that location of link_object to “true”
--
--testLink is used to see if we’ve already been to a page. It indexes
--into link_object the same way as visitLink and returns whatever is in
--that slot.
--
--hidden is the variable that has the status of each pages’ annotation
--window. If a pages window is hidden, then item (the number of that
--page) of hidden will be true
--
--start_time is used by the close project function to compute total usage
--by the student.
--
--Initialize project variables.
on openProject
global link_object, link_distance, hidden, usage, usage_num,
start_time, name_string
put seconds() into start_time
put 0 into link_distance
put 0 into tracker_num
repeat with COUNT = 1 to the number of cds in this proj
put “false” into item COUNT of link_object
put “true” into item COUNT of hidden
end repeat
ask “What is your name (first and last)?”
put it into name_string
end openProject

on visitLink link_name
  global link_object
  put “true” into item (the number of cd link_name of this proj) of
        link_object
end visitLink
function testLink link_name
    global link_object
    if item (the number of cd link_name of this proj) of link_object is
        "true"
        return "true"
    else
        return "false"
    end if
end testLink

on closeProject
global name_string, usage, start_time
if name_string is empty then exit closeProject
put name_string&tab&(seconds()-start_time)&return into temp
put usage after temp
put the short date into fName
put name_string after fName
savefile fName temp
end closeProject

-- NOTE the log path is hard wired. This is easy to change, but there’s no
-- need right now
on savefile fName,temp
global shellPath
put shellPath&"Adam Projects:Knee Folder:Student use logs:"&fName into
    fName
open file fName
write temp to file fName
close file fName
end savefile

C.2 Standard Knee Page Scripts

on openCard
global link_object, link_distance, link_stack, hidden, time

-- start the time to keep track of how long users on this page
put seconds() into time

-- prevent any screen modifications or activity
lock screen

-- push this card onto the backtrack stack
put link_distance + 1 into link_distance
put (the short name of me) into item link_distance of link_stack

-- mark this card as visted (for link coloring)
put "true" into item the number of me of link_object

-- check links and color appropriately
repeat with COUNT = 1 to the number of cd grcs
put (the short name of cd grc COUNT) into grc_name
if (word 1 of grc_name is “link”) then
if testLink(word 2 of grc_name) is “true” then
set fillFore of grc COUNT of this cd to 7
else set fillFore of grc COUNT of this cd to 19
end if
end repeat

-- Hide the annotate field
if item (the number of me) of hidden is “true” then hide cd field
annotate
unlock screen
end openCard

on closeCard
  global usage, time
  -- update the usage timer
  put (the short name of me)&tab&(seconds()-time)&cr after usage
end closeCard

C.3 Standard Knee Link

on mouseUp
  global moduleView, curInk, adamShell, saveFlag
  put “true” into saveFlag
  setWindow topWindow()
  send lockUp to proj adamShell

  -- Let the cursor be something other than an arrow.
  set lockCursor to “false”

  -- Jump to the card pointed to by this link
  go cd (word 2 of the short name of me) of wd 1 of this proj
  unlock screen
end mouseUp

-- These two make the cursor an arrow when it’s over the link. By holding
-- down the shift key and moving into the link, when Balloon help is on,
-- you’ll get a message. This is sneaky because the current page names are
-- not meaningful to the average user (the names aren’t the same as the
-- page titles at the top of the screen).

on mouseEnter
  set lockCursor to “true”
  set cursor to arrow
  if the shiftKey is “down” then
    FullBalloons “ShowDirect”, “Jump to page ”&(word 2 of the short name of me)
  end if
end mouseEnter
on mouseLeave
    set lockCursor to "false"
    set cursor to hand
end mouseLeave

C.4 Menu Button

on mouseUp
    global moduleView, curInk, adamShell, saveFlag
    put "true" into saveFlag
    setWindow topWindow()
    send lockUp to proj adamShell
    go cd Knee_Menu of wd 1 of this proj
    unlock screen
end mouseUp

on mouseEnter
    set lockCursor to false
    FullBalloons "ShowDirect","Go to the Main Menu"
end mouseEnter

C.5 Backtrack Button

-- Pop me off the link stack, go to the page before me, pop it off the
-- link stack since it will be added when it opens (and we don’t want it
-- in the stack twice in a row.

on mouseUp
    global link_distance, link_stack
    if (link_distance < 2) then exit mouseUp
    put (link_distance - 1) into link_distance
    lock screen
    go to cd (item link_distance of link_stack)
    put (link_distance -1) into link_distance
    unlock screen
end mouseUp

on mouseEnter
    set lockCursor to false
    FullBalloons "ShowDirect","Go back the way you came. Repeatedly hitting
    back will take you to the beginning of the module."
end mouseEnter

C.6 Annotate Button

on mouseUp
    global hidden
    if item (the number of this cd) of hidden is "true" then
        put "false" into item (the number of this cd) of hidden
        show cd field annotate
    else
        put "true" into item (the number of this cd) of hidden
end mouseUp
hide cd field annotate
choose browse tool
end if
end mouseUp

C.7 Help Button

on mouseUp
  global moduleView,curInk,adamShell,saveFlag, helpPage
  put "true" into saveFlag
  setWindow topWindow()
  send lockUp to proj adamShell
  put the short name of this cd into helpPage
  go cd Help of wd 1 of this proj
  unlock screen
end mouseUp

on mouseEnter
  set lockcursor to false
  FullBalloons "ShowDirect","Go to the Help pages."
end mouseEnter

C.8 Programmer’s Page Script

on openCard
  show cd grc cover
  answer "Did you really mean to get to this page?" with "Yes" or "No"
  if it is "No" then
    go cd "Knee_Menu" of this proj
    exit openCard
  end if
  ask "What is the password"
  if it is "dana779" then
    hide cd grc cover
    else
    go cd "Knee_Menu" of this proj
  end if
end openCard

C.9 View Test File (on programmer’s page)

on mouseUp
  global shellPath
  select text of cd field viewer
  type "

  ask “What is the name of the student (first last)?”
  put it into file_name
  ask “What date (mm/dd/yy)?” with the short date
  put it before file_name
  put "Opening file "&file_name&cr into msg
show msg
put shellPath&"Adam Projects:Knee Folder:Student use logs:"&file_name into file_name
open file file_name
repeat
read from file file_name for 16834
if It is empty then exit repeat
put It after tmp
end repeat
close file file_name
hide msg
put tmp into cd field viewer
show cd field viewer
end mouseUp

C.10 Clear Link Colors (on programmer’s page)
on mouseDown
  global link_object
  if the shiftKey is "down" then answer the short name of me
repeat with COUNT = 1 to the number of cds
  put "false" into item COUNT of link_object
end repeat
  send "openCard" to this cd
end mouseDown

C.11 Edit link/page (on programmer’s page)
on mouseUp
  answer “Replace page scripts or link scripts?” with “Page” or “Link” or “Cancel”
  if it is “Cancel” then exit mouseUp
  if it is “Page” then
    -- ITS A PAGE
    ask "Starting card" with "1"
    put it into scard
    ask "Ending card" with the number of cds
    put it into ecard
    answer "Is the replacement script in this page’s text field?” with “Yes” or “No”
    if it is “No” then
      answer “Put the replacement script in the text field”
      exit mouseUp
    end if
    repeat with count = scard to ecard
      set the script of card count to cd field viewer
    end repeat
    answer “Done”
    exit mouseUp
  end if
  -- ITS A LINK

124
answer "Is the replacement script in this page’s text field?" with "Yes" or "No"
if it is "No" then
answer "Put the replacement script in the text field"
exit mouseUp
end if
repeat with count = 1 to the number of cds
repeat with nl = 1 to the number of grcs of cd count
if word 1 of the short name of grc nl of card count is "link" then
set the script of grc nl of card count to cd field viewer
end if
end repeat
end repeat

answer "Done"

end mouseUp

C.12 Clear Link Stack (on programmer’s page)
on mouseUp
  global link_distance, link_stack
  put 1 into link_distance
  put "Knee_Menu" into item 1 of link_stack
end mouseUp

C.13 List Link Stack (on programmer’s page)
on mouseUp
  global link_distance, link_stack
  answer link_distance
  select text of cd field viewer
  type " "
  repeat with COUNT = 1 to link_distance
  put item COUNT of link_stack into line COUNT of temp
  end repeat
  put temp into cd field viewer
end mouseUp

C.14 Quiz Button
on mouseUp
  global shellPath
  put shellPath&"Adam Projects:Knee Folder:quiz.questions" into file_name
  open file file_name
  put 1 into i
  repeat
  repeat with j=1 to 4
read from file file_name until ","
if It is empty then exit repeat
put It into item j of line i of questions
-- read from file file_name for 1
end repeat
put j+1 into j
if it is empty then exit repeat
read from file file_name until return
put It into item j of line i of questions
put i+1 into i
end repeat
close file file_name

put 0 into score

answer "Type quit at any time to quit."&cr&"or click cancel to skip a
question"
put i-1 into j
repeat with i = 1 to j
put "false" into done
put item 1 of line i of questions into temp
show cd grc temp
show cd grc (item 2 of line i of questions)
ask item 3 of line i of questions
repeat until done is "true" or done is "exit"
if it is "quit" or it is "Quit" or it is "QUIT" then put "exit" into
done
if it is item 4 of line i of questions or it is item 5 of line i of
questions then
answer "Right On!"
put score + 1 into score
put "true" into done
end if
if it is "" then put "true" into done
if done is "false" then ask "Sorry, try again :"
end repeat
hide cd grc item 1 of line i of questions
hide cd grc item 2 of line i of questions
if done is "exit" then exit mouseUp
end repeat
answer "Your score was "+score+" out of "+j
end mouseUp
Appendix D

Instructions for the Template Module

1. First, you’ll probably want to figure out how many dissection/atlas pages you need. It might be good to scan pictures, enter text, etc., now so that all you need to do is put it together.

2. Copy the module named “TEMPLATE” and rename it to whatever the subject matter of the new module is (e.g. ARM)

The template module assumes that you are going to have:
   a) A text based menu
   b) Intro pages
   c) Dissector pages
   d) Quiz page
   e) Atlas pages
   f) Help pages
   g) Programmers page

There is one template page for each of these sections. Any of these pages can be deleted without side effects except for the menu page and help page. If you want to delete either of these pages, then you also need to delete all of the menu and/or help buttons (so delete them from the template pages before continuing).

3. Go to the Edit menu and copy and paste each page as many times as you need. If you need 10 atlas pages, paste the atlas template page 9 times (+1 original = 10).

4. Give each page a unique name by going to the Info menu, selecting page info, and then typing in a one word name (use “_” as virtual spaces so “knee menu” becomes “knee_menu”). At some point, someone can change things so page names can be more than one word.

5. Update each page’s info (e.g. like the page titles, text, pictures). Text can be copied and pasted from other sources (e.g. Word) and pictures can be imported (under the file menu).
6. Copy and paste link objects over the appropriate words in text fields. Change the link names so that the name is “link” a space, and then the page name that the link jumps to (NOT the page title). So a link that jumps to a page named Knee_menu should have the name “link Knee_menu”.

7. Update the menu. To do this, you have to type the title of every page into the text field on the menu page. You then need to make one link for each page in the menu. Since the links have to use page names rather than titles, it might be best if you go to the programmers page, list the page names, create the links, and then copy them back to the menu page en masse. To find out where a link goes, turn on balloon help, hold down the shift key and then move into the link in question. A balloon with the page name will appear. Take a look at the Knee module’s menu to get idea about how you might want to format things. At some point, someone may want to make a graphical template to replace the text menu.

8. Reposition the annotation fields if necessary (hopefully, at some point someone will replace the annotation fields with an entire page snapshot so this won’t be necessary).

9. Edit the scripts of graphic objects that you want to be clickable. Unfortunately, you will have to figure out what region boundaries are by holding down the option key and clicking on the picture. It will then return x & y coordinates that you can use to define a region boundary in the graphic’s script. The graphic object on the intro page has a sample script that can be copied and pasted into other objects. If you want regions to display something other than the area name, update the list at the beginning of the script.

10. At this point, just about everything should be completed, except that you may want to move text, pictures, and links around, and update the quiz file. When moving many objects, you can hold down the shift key when clicking, and this will allow you to select more than one object. Then moving one of the selected objects will move them all. Updating the quiz file requires that you create sets of two pictures and a question about the pictures. Read the section in Chapter 6 to get an idea of how the quiz facility works. You should then edit the script of the “Make Quiz File” button (on the programmers page) to make your file. The script of that button explains what you should do.

11. Note that any data files are currently saved in the knee folder. You should change the project and programmers page scripts to save in the appropriate folder.
References


