Further Development of Subsurface Profiling and Engineering Geology Software

by

Jennifer Susan Hango

B.S., Individual Program of Studies in Geology, Harvey Mudd College (1995)

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

© Massachusetts Institute of Technology 1998. All rights reserved.

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

Department of Civil and Environmental Engineering

May 22, 1998

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

Herbert H. Einstein

Professor, Department of Civil and Environmental Engineering

Thesis Supervisor

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

Joseph M. Sussman

Chairman, Departmental Committee on Graduate Studies
Further Development of Subsurface Profiling and Engineering Geology Software

by

Jennifer Susan Hango

Submitted to the Department of Civil and Environmental Engineering on May 22, 1998, in partial fulfillment of the requirements for the degree of Master of Science in Civil and Environmental Engineering

Abstract

The Physical Geology Tutor, Nomad, and the Kriging-Bayesian (KRIBS) module are educational geoscience software that were developed at M.I.T. The Physical Geology Tutor is an interactive tutorial program that helps students learn about physical geology. It consists of eight chapters that are comprised of three components each: a Textbook, a Quiz, and a Workbook. In this thesis, the Physical Geology Tutor was transformed to run on the World Wide Web. Nomad/KRIBS is a three-dimensional subsurface modeling program written in C++. It creates simple cross sections by allowing users to draw ground layer boundaries between consecutive boreholes, and it also creates subsurface profiles using an Indicator CoKriging model with Bayesian updating.

This thesis describes the further developmental work done with the Physical Geology Tutor and Nomad/KRIBS. A description of the Physical Geology Tutor and its new format is included, as well as explanations of some new features. An in-depth discussion of the Workbook section of the Tutor is also presented. A discussion of added features in Nomad/KRIBS is included, with descriptions of a two-dimensional semi-variogram surface model and a nested semi-variogram model.

Thesis Supervisor: Herbert H. Einstein
Title: Professor, Department of Civil and Environmental Engineering
Acknowledgements

I would like to offer sincere thanks to my advisor, Herbert Einstein, for his support, encouragement, advice, and especially patience over the last three years.

I would also like to thank Patrick Kinnicutt for his suggestions for the Physical Geology Tutor, for his assistance with and suggestions for Nomad/KRIBS, and for being something of a mentor for this geologist-turned-engineer-turned-programmer.

I'd like to thank my UROPs Alice Yang and Baffour Abedi-Asante for their work with the PGT Workbook. Thanks are also extended to Jeff Allen and Richard Stallman for their helpful suggestions for the layout of the Textbook pages, and to Reid Pinchback for his help with the initial design of the PGT Workbook applets.

Je voudrais remercier Claude Indermitte de ton aide avec des problèmes de matériel, des questions au sujet de C++, des ordinateurs en général, la prononciation et la conversation françaises, m'encourager à apprendre faire du patin à roulettes et de glace, et d'être un ami depuis les trois années dernières.

I'm glad to call a number of students from Courses 1 and 6 friends; these people were great to talk to, have dinner with, play jokes on and with, and to just be around. I'd like to say thanks for being there to Antonio Bobet, Violeta Ivanova, and the 1.00 T.A.s from Fall 1997 and Spring 1998, especially Andrew Walsh and Larry Chang, the other two members of the 1.00 All-Nighter Club.

I'd like to thank Cliff McCarthy for his friendship, support, encouragement, assistance with computer questions, and all the little things that are too numerous to list here.

I'd like to thank Mary Agner for being a friend, a confidante, a shoulder to cry on, and an all-around great person. These words aren't terribly poetic, but the point is made.

I especially want to thank my parents, Joe & Susie Hango, for their love, support, and encouragement throughout the years, and especially during my time here at M.I.T.

Please to remember, there is No Stopping Any Time....
## Contents

1 Introduction  

2 The Physical Geology Tutor  
   2.1 Introduction  
   2.2 PGT Background  
   2.3 Textbook  
   2.4 Quiz  
   2.5 Workbook  
   2.6 Recommendations for Future Work  

3 PGT Workbook — Java Code  
   3.1 Introduction  
   3.2 The Java Language  
   3.3 The Applet  
      3.3.1 Naming Classes  
      3.3.2 Outlining Classes  

4 Nomad/KRIBS  
   4.1 Introduction  
   4.2 Nomad/KRIBS Description  
      4.2.1 Nomad/KRIBS History  
      4.2.2 Nomad/KRIBS Program Features  
   4.3 Semi-Variogram Modeling  
      4.3.1 The Semi-Variogram
4.3.2 GSLIB and UNCERT Packages ........................................ 48
4.3.3 KRIBS Implementations .............................................. 49
4.4 Recommendations for Future Work .................................. 53

5 Conclusion ......................................................................... 57
  5.1 Contributions .............................................................. 57
  5.2 Recommendations ........................................................ 58

A HTML Physical Geology Tutor Administrator’s Manual .......... 59
  A.1 Introduction ............................................................... 59
  A.2 Unix Background ........................................................ 60
    A.2.1 Accessing the /mit/1.38 and /mit/1.38_dev Project Athena Lockers
                 ................................................................. 60
    A.2.2 Starting Netscape .................................................. 60
    A.2.3 Copying Files in Unix ............................................. 61
    A.2.4 Using Emacs ......................................................... 63
    A.2.5 Using XV ............................................................. 63
  A.3 PGT Resources ........................................................... 68
    A.3.1 Image Locations .................................................... 68
    A.3.2 Obtaining Digitized Images ...................................... 69
    A.3.3 pgt-admin@mit.edu ............................................... 69
    A.3.4 Finding Help ......................................................... 70
  A.4 Textbook .................................................................... 70
    A.4.1 HTML Files ........................................................... 70
    A.4.2 Editing and Adding Textbook Files ............................... 72
  A.5 Quiz ........................................................................... 74
  A.6 Workbook ................................................................... 78
    A.6.1 HTML Files and Perl Script ........................................ 78
    A.6.2 Java Source Code and Class Files ................................ 80
  A.7 Index .......................................................................... 82
  A.8 Bringing It All Together ............................................... 83
  A.9 Restarting the httpd Server .......................................... 86
E.3 Outline.java ......................................................... 134

F Workbook Page Generation Script Code 147

Bibliography 155
List of Figures

2-1 The Title Page of the Physical Geology Tutor ......................... 20
2-2 A Textbook Page from the Physical Geology Tutor .................... 22
2-3 A Chapter Table of Contents Page ................................... 23
2-4 Navigation Bar Used in the Physical Geology Tutor .................. 23
2-5 Text Part of a Textbook Page ......................................... 24
2-6 Textbook Page with Text and Illustrations .......................... 25
2-7 A Single-Answer Quiz Page ............................................ 26
2-8 Name Only Workbook Page (Horizontal Orientation) ................. 27
2-9 Name Only Workbook Page (Vertical Orientation) .................... 28
2-10 Outline Only Workbook Page (Horizontal Orientation) .............. 29
2-11 Outline Only Workbook Page (Vertical Orientation) ................. 30
2-12 Name and Outline Workbook Page (Horizontal Orientation) ........ 31
2-13 Name and Outline Workbook Page (Vertical Orientation) .......... 32
3-1 Template for a Horizontal Layout Workbook Page ................... 38
3-2 Template for a Vertical Layout Workbook Page ....................... 39
4-1 Nomad Main Window with Example Data File .......................... 45
4-2 Experimental and Exponential Model Variograms ...................... 46
4-3 Ground Profile Obtained by Using KRIBS ............................ 47
4-4 Basic Principles of and Parameters Used for Calculating Directional Variograms 50
4-5 18 Directional Variograms Plotted on Same $\gamma(h)$ vs $h$ axis. The legend of symbols is shown to the right of the graph; each symbol represents a directional variogram. .......................... 51

11
Chapter 1

Introduction

Because of ever-changing hardware and software technology, it is necessary to update programs regularly to ensure that they operate in an efficient manner and do not become legacy software that runs only on obsolete systems. Both the Physical Geology Tutor and Nomad have gone through numerous changes and upgrades over the past 10 years. Sometimes the changes are necessitated by an older technology becoming obsolete, and sometimes because a new technology is better-suited for the task at hand. Additional enhancements that are somewhat technology independent may also prompt a rewrite or upgrade. In some cases, two or more of these factors can come into play.

The previous version of the Physical Geology Tutor (PGT) was written in AthenaMuse and required specialized hardware and software for its use. A new and incompatible version of AthenaMuse was introduced, and the original version was no longer supported by Project Athena. Support for the PGT became increasingly difficult, and recoding the Tutor became a priority. HTML and the World Wide Web (WWW) were chosen for the new Tutor partly because of the author's familiarity with them, and partly because a Web format would ensure that anyone at M.I.T. with a computer and a graphics-based web browser would be able to access the Tutor, rather than having to use one specific machine. The availability of a method for checking Quiz answers and the introduction of Java also made an HTML version of the Tutor a feasible option.

Nomad's recent changes have largely been enhancements in functionality. Kinnicutt
rewrote Nomad in ANSI\textsuperscript{1} C++ and added a major component that employs Indicator CoKriging algorithms to quantitatively determine the composition of the subsurface by ground type [Kinnicutt, 1995]. More recently, Kinnicutt replaced the directional kriging code with a function that would solve the entire system. For this thesis, it was decided to focus efforts on modifying the code that deals with semi-variogram modeling.

Chapter 2 contains a description of the HTML version of the Physical Geology Tutor. The different components of the PGT are discussed, as well as additions to the PGT and differences from previous versions. Several recommendations for future work are also given at the end of the chapter.

Chapter 3 is an in-depth description of the Java applet used in the PGT Workbook exercises. A short history of Java is presented, as well as a discussion of the algorithms and Java classes used.

Chapter 4 contains a description of the additions made to the Nomad/KRIBS ground profiling program. The chapter begins with a brief history of Nomad and KRIBS and an overview of the program's functionality. It then describes the functions used for creating two-dimensional semi-variogram surfaces and for calculating nested variogram models.

Chapter 5 summarizes the contributions and recommendations made in this thesis.

Appendix A is a comprehensive guide for administering the Physical Geology Tutor. It describes how to create and edit new Textbook, Quiz, and Workbook pages. Additionally, it provides basic instructions for using other applications that are useful in creating the new files. Instructions for managing the webserver that serves the Tutor are included, as well as a list of sources for help on the M.I.T. campus.

Appendix B is the User's Manual for the Physical Geology Tutor. It is intended to be a reference for students and others who use the PGT as a learning aid. It provides descriptions of the different components of the PGT and instructions for using each component. It also contains a list of common problems that may be encountered while using the Tutor. This Appendix is also published as a standalone document that is distributed in the M.I.T. 1.32 and 1.38 Engineering Geology courses.

Appendices C-F are source code listings. Appendix C contains example Textbook and

\textsuperscript{1}American National Standards Institute
Quiz files for the PGT, and Appendix D contains the template Textbook and Quiz files which should be used when creating new Textbook and Quiz pages. Appendix E lists the source code for the three Workbook applets: Both.java, Name.java, and Outline.java. Appendix F is the source code for the Perl script used to generate the HTML files used in the Workbook exercises.
Chapter 2

The Physical Geology Tutor

2.1 Introduction

The Physical Geology Tutor is an interactive computer-based educational tool used in the Massachusetts Institute of Technology's 1.32 and 1.38 Engineering Geology classes. It is a study aid for students who are learning about physical geology, and it is meant to augment lab and lecture material. Students can read about geologic features and processes in the Textbook section, use the quiz section to test their understanding of the concepts they have just learned, and check their ability to recognize and name geologic features in the Workbook section. The Title Page of the Physical Geology Tutor is shown in Figure 2-1. The three major Tutor sections will be discussed below, including more detailed descriptions of the functions of each section, as well as changes from and additions to each section from older versions of the PGT. Possible future additions and changes to the PGT are also described at the end of the chapter. Documentation for the Tutor is included in Appendices A and B, and includes a manual for administering the PGT and a user's manual.

2.2 PGT Background

Einstein outlined the structure of an Engineering Geology Educator (EGE) [Einstein, 1986]. The Educator was to be composed of 3 modules: a geology tutorial program, a geologic assessment and exploration program, and an engineering consequences and assessment program. Donald Gray [Gray, 1987] implemented the first module of the EGE in the M.I.T.
Physical Geology Tutor

HTML Version 1.0
Department of Civil and Environmental Engineering
Massachusetts Institute of Technology

Note: You may need to resize your browser window to see the entire page properly.

- Using the Physical Geology Tutor
- Eolian Chapter
- Fluvial Chapter
- Glacial Chapter
- Karst Chapter
- Lacustrine Chapter
- Shoreline Chapter
- Structural Chapter
- Vulcanism Chapter
- PGT Main Index
- Photograph and Illustration References
- Questions, Comments, Ideas
- PGT Survey

Figure 2-1: The Title Page of the Physical Geology Tutor
developed General Engineering Problem Solving Environment (GEPSE). This tutorial program had a Reference section which described geologic features and presented pictures of them, and a Quiz section which asked questions about the material presented in the Reference section. The Physical Geology Tutor was written by Carlos Noack [Noack, 1988] in 1987-1988 for use on Athena\(^1\) computers running the X10 windowing platform. Noack’s Tutor was comprised of a Textbook section, a Quiz section, and a Workbook section. The Workbook section displayed images of geologic features, and students needed to identify the feature, either by naming it, drawing an outline around it, or doing both actions. Later, when Athena started to use the X11 windowing system, Matthew Grebner [Grebner, 1989] wrote a new version of the PGT. This new version of the Tutor made use of a new windowing interface and programming language developed at M.I.T. called AthenaMuse. Eight distinct Textbook chapters were introduced along with corresponding Quiz sections. An Index was added for each chapter. Patrick Kinnicutt [Kinnicutt, 1991] further developed the Tutor, and ported it to private workstations when Athena stopped supporting the original version of AthenaMuse and the hardware needed to run the Tutor. Kinnicutt’s Tutor had an expanded workbook section, encompassing seven of the eight Textbook chapters, and it reimplemented the polygon-drawing function. A Look-Up feature was also added to the Tutor; students could select certain words on Quiz and Workbook pages and look up the Textbook entries for those words.

2.3 Textbook

The Textbook section of the Physical Geology Tutor defines, describes, and illustrates geologic terms and concepts. Figure 2-2 shows a typical Textbook page. All Textbook pages fall into one of eight different chapters: eolian, fluvial, glacial, karst, lacustrine, shoreline, structural, or vulcanism. All the Textbook pages for a chapter are listed on a chapter Table of Contents page. An example of a Table of Contents page is shown in Figure 2-3. Students can click the mouse on any Textbook page name listed to go directly to that page. A chapter Index, which lists all the geologic terms used in the chapter, can also be used to

\(^{1}\)Originally Project Athena. Athena is the M.I.T. Unix network.
8.4.7 Pahoehoe Lava

Pahoehoe or ropy/corded lava is a mass of lava which has solidified so that its surface is smooth and exhibits ropy or corded shapes.

Pahoehoe lava oozing out at edge of a lava lake in Kilauea Iki, Hawaii

Original picture from the Ward Collection

Figure 2-2: A Textbook Page from the Physical Geology Tutor
access individual Textbook pages. Students can click on a term in the Index to go to the page where the term is first used or defined. A third way to access individual pages is to use the Navigation Bar located at the top and bottom of a page (Figure 2-4). Students will click on Next to go to the next page and Prev to go to the previous page.

Once a student selects a Textbook page, the page will load and appear in the web browser window. All Textbook pages have text describing a particular geologic feature or concept. An example of the text portion of a Textbook page can be seen in Figure 2-5. The geologic term, any synonyms, and other notable geologic terms are shown in boldface text. Notable terms which are defined on Textbook pages elsewhere in the Tutor (both in the same chapter as well as in other chapters) are hyperlinked to those pages and shown in blue underlined text, meaning that the student can click the mouse on the hyperlinked term and
2.4.1.3 Rapids

Rapids occur where hard rock layers or debris such as boulders cause the gradient to locally increase downstream. Rapids are similar to waterfalls except the gradient is not large enough to cause plunging. The turbulence through the rapids causes the erosion.

Water approaching the rapids becomes smooth and less turbulent; this condition is called shooting flow. At some point, the thickness of the water increases and the zone of great turbulence begins; the beginning of this zone is called the hydraulic jump. Boulders, cobble, and pebbles swirling in the turbulent water often erode cylindrical hollows which are called potholes or scour pockets.

Figure 2-5: Text Part of a Textbook Page

go directly to the page where the term is defined. This feature is new in the HTML version of the PGT, and it allows students to quickly and easily look up unfamiliar geologic terms.

Many Textbook pages have, in addition to the text, drawings and photographs to illustrate geologic features and processes. The images are displayed below the text, and a short description of the illustration and a credit for the picture source are given. Small, “thumbnail” versions of the images are displayed below the large image on a page. If more than one image is available on a particular page, the student can click on the thumbnail images to see a larger version of the pictures. An example of this type of Textbook page can be seen in Figure 2-6.

2.4 Quiz

Each chapter in the Physical Geology Tutor has a Quiz component associated with it. Students can use the Quizzes to check their knowledge of topics presented in the PGT. The Quizzes, ranging in length from 10 to 230 questions per chapter, are indexed at the beginning of every Quiz section, so students can easily come back to a specific question after doing other activities. The Quiz questions are also sequentially linked in the Navigation Bar.
3.3.1.4 Horn

A horn is a single remnant or high point of rock which forms when three or more cirques converge from different directions.

The Matterhorn.
Original from the Ward Collection
so that students can go on to the next question by clicking on the word Next (Figure 2-7).

Figure 2-7: A Single-Answer Quiz Page

The Quiz questions are multiple choice; a student selects her answer and marks her choice in checkboxes. She then submits her answer, and the PGT checks the answer for correctness. If the answer is correct, the PGT tells the student that she is correct, and she can advance to the next question. If the student submits an incorrect answer, the PGT tells her the answer is wrong and prompts her to submit a new answer. The PGT does not display correct answers on request; students must submit answers themselves to determine if they are correct. Some questions have more than one correct answer; students submit one answer at a time and the PGT tells them if each answer is correct or not.

The Quiz pages are CGI\(^2\) scripts written in the Perl programming language. They generate the HTML code that the web browser uses, as well as perform the answer checking. The code for a sample page is included in Figure A-16.

\(^2\)Common Gateway Interface. This is a way to let web pages execute programs on the server computer.
2.5 Workbook

Each chapter in the Physical Geology Tutor except the Lacustrine chapter has an associated section. The Workbook section displays an image of a geologic feature, and students are required to either name the given feature, outline the given feature, or both name and outline the feature.

The Name Only section displays an outlined image and a series of Text Entry/Display fields and clickable buttons (Figures 2-8 and 2-9). Students type the name of the feature in the top left text entry field and click on the Submit Answer button with the mouse. The PGT compares the student’s answer with the correct answer. If the student is correct, “You’re right!” is displayed beneath the student’s answer. If the student is incorrect, she is told that her answer is wrong and that she should try again. After a student has submitted
Figure 2-9: Name Only Workbook Page (Vertical Orientation)
an answer with the Submit Answer button, he can display the correct answer by clicking on the Show Answer button. The correct answer will be shown underneath or next to the word “Answer:”. At any time, students can click on the button marked Clear Text. This resets the exercise and clears all text from the text boxes, so a new answer will need to be submitted in order to be able to use the Show Answer button.

The Outline Only section displays an unoutlined image and a series of text display fields and clickable buttons (Figures 2-10 and 2-11). The name of a geologic picture is displayed, and the students must draw an outline around the named feature in the picture. Students draw an outline by clicking the mouse on the image to set the vertices of a polygon. When the mouse is moved, a line segment is drawn between the last placed point and the mouse pointer. The line segment changes color and no longer moves when a new vertex is placed. Students must close their polygons in order for the PGT to be able to check their outlines;

Figure 2-10: Outline Only Workbook Page (Horizontal Orientation)
Figure 2-11: Outline Only Workbook Page (Vertical Orientation)
to close the polygon, they must click the mouse on or near the initial vertex. Once the student has finished drawing the polygon, she should press the Check Outline button for the Tutor to check the outline. Students are told if their answers are correct or incorrect. If the student is incorrect, she is advised to press the Clear Outline button and try again. The Clear Outline button may be pressed at any time; the user needs to move the mouse pointer back onto the image to make the outline disappear. After an outline has been submitted, the student also has the option of displaying the correct outline. This feature can only be used after an outline has been submitted, and the user must move the mouse pointer back onto the image to display the correct outline.

The Name and Outline section combines both of the previous exercises. Students are shown an unoutlined image, and the text fields and push buttons used in the Name Only and Outline Only sections are displayed (Figures 2-12 and 2-13). Students must provide
Figure 2-13: Name and Outline Workbook Page (Vertical Orientation)
both the name of the feature and an outline around the feature. These two parts of the exercise can be done independently of each other; students can submit either the name or the outline first. It is even possible to type a partial word or outline and then switch to the other part of the exercise.

The Workbook pages all display the references for the photographs used in the exercise. All the pages also have a “Lookup” function. If a student wants to go back to the associated Textbook page to read more about a geologic feature, she can click on the Look up feature link and the Textbook page will be loaded in the browser window. To return to the Workbook, she should use the browser’s Back button. Using the Prev link would take her to the Textbook page that immediately precedes the current page in the Table of Contents, rather than return her to the Workbook page she was using. The Workbook pages use the PGT Navigation Bar, so students can move through the Workbook sequentially. The Workbook starts with the more difficult Name and Outline section, and then progresses to the easier Name Only and Outline Only sections.

2.6 Recommendations for Future Work

Several additions and changes could be made to the Physical Geology Tutor to enhance its usefulness to students. Animated segments, additional chapters, and virtual field trips are additions from which students could benefit.

Because the Tutor is available on the World Wide Web, animated sequences can be added to Textbook pages. Several Java applets exist which allow programmers to display a series of pictures in quick succession. Some of these applets are “ready to use”; all that the PGT Administrator would need to do is provide the names of the required images in an HTML file. It is also possible to make animated GIF\(^3\) files. Programs which create animated GIFs take a series of GIF images and merge them into one file; the individual images are displayed in quick succession. Once the new image is created, no special coding beyond the \(<\text{img}>\) HTML tag is needed to display it. Geologic processes such as the slip motion on a fault or diapiric migration of salt or mud could be shown. Video footage can be

\(^3\)Graphics Interchange Format
digitized and stored as MPEG\textsuperscript{4} or QuickTime files; footage of live-action geologic processes, such as volcanic eruptions, could be presented.

New chapters could be added to the PGT. A chapter that briefly describes mineralogy could be useful to students who are unfamiliar with common minerals such as quartz, feldspar, and hornblende. A new chapter on mass wasting would also be a worthy addition. Students would have an on-line reference for the downslope movement of material. Other applied chapters, such as those suggested by Grebner [Grebner, 1989], would be useful additions.

The M.I.T. Department of Civil and Environmental Engineering’s Engineering Geology classes (1.32 and 1.38) incorporate field trips into the curriculum. There is no substitute for going to an outcrop or other geologic feature and observing and studying it firsthand, but if a student is ill on the day of the field trip or wants to review the features seen at a particular site, a virtual field trip could be a helpful resource. Poor weather conditions, foliage, and construction may obscure or destroy completely some features, and photographs may be the only way for students on a particular field trip to observe certain features. Pictures and text descriptions of the field trip’s geologic features would need to be compiled and transferred to electronic formats. Many institutions have webpages for virtual field trips, and the PGT could include a list of some of the better of these pages. Students could use these pages to learn about geology outside the Greater Boston region.

A limitation of Workbook section is that the student must type the name of a geologic feature exactly as it appears in the answer string; if the student capitalizes a word or adds an “s” at the end of a term, the applet considers the student’s answer to be incorrect. Code libraries exist for text strings called “regular expressions,” or “regexps,” which match patterns. Regexps could be used to match minuscule or majuscule characters at the beginning of words and check for an extra “s” at the end of a word. Students who capitalize or pluralize words would then be told their answers were correct, even though their answers are not a strict character-by-character match with the answer string.

\textsuperscript{4}\textsuperscript{Moving Picture Expert Group. MPEG is a type of animation file which many web browsers can process.}
Chapter 3

PGT Workbook — Java Code

3.1 Introduction

Java is the *wunderkind* of WWW programming languages; introduced to the programming community in 1995, it allows programmers to write interactive, multimedia programs, called applets, that can be run from any web browser with Java capabilities on any type of computer, regardless of where the applets were written or compiled. When rewriting the PGT into a web-based format, Java was the logical choice for implementing the Workbook section. This chapter begins by providing some background information about the Java programming language. It then provides a description of the workings of the three applets used in the Workbook section, followed by detailed explanations of the outlining and naming functions used.

3.2 The Java Language

Java is an object-oriented, C++-based programming language that was developed by Sun Microsystems in the early 1990s. It is a purer object-oriented language than C++, is more compact than C++, and dispenses with explicit pointers entirely. Most data types, with the exception of the primitive data types (including *boolean* and *arithmetic* types), are dynamically allocated, and references to instances of those types (objects) are handled with implicit pointers.

Java programs are compiled to a byte code state, which means that a second "inter-
preter" program is needed to execute the programs. Compilers produce class files, which may then be run on a Java Virtual Machine (JVM). The Java Virtual Machine is an abstraction of a generic computer; the Java class files are designed to run on this generic machine rather than on a specific type of computer with a specific type of operating system. Many different computer architectures have a Java Virtual Machine; the JVM serves as a bridge between different architectures and the “Write Once, Run Anywhere” Java class files. This is extremely useful, because Java class files do not need to be recompiled to run on different machines. A Java class file produced on a computer running Solaris can be run on a machine running HP-UX, Linux, WindowsNT, MacOS, or another operating system, as long as each of those machines has a JVM for the specific architecture of the machine.

Applets are special types of Java applications. Applets are small programs that can be run on a webpage by a Java-compatible browser. Newer versions of Netscape Navigator/Communicator, Sun HotJava, and Microsoft Internet Explorer are among the browsers that have a Java Virtual Machine included. When an applet is embedded on a webpage, the computer accessing the webpage downloads a copy of the applet and runs the applet on its hard drive. For this reason, security issues are important to consider, and Java applets have very specific functions they are not allowed to do, such as writing files on the computer running the applet and executing system commands.

Applets have the ability to read parameters from the HTML file. The advantage of passing parameters in from the file that calls the applet is that these parameters do not need to be hardwired into the source code. This allows the user to specify certain attributes about the applet (such as color, images to display, coordinates of a polygon, etc.) without needing to recompile the applet for each specific case. This feature also means that the user of the applet class files does not need to be concerned about the inner workings of the applet source code; he can simply change the parameters to other legal values and be assured that the applet will function as anticipated.

### 3.3 The Applet

Applets are an extension of the base class `java.applet.Applet`; they inherit their basic functionality from this class. Secondary classes are often written to provide specialized func-
tionality for a specific applet. In the Workbook applets, the Applet class file is modified, and separate classes for the data input and output provide the methods (functions) for displaying graphics, drawing outlines, and handling text input and output. The Both.class file is used by the Name and Outline Workbook applet, the Name.class file is used by the Name Only applet, and the Outline.class file is used by the Outline Only applet. The source code for the Workbook applets is listed in Appendix E.

Depending on the RAM availability and speed of the computer downloading an applet, it may take several seconds to a few minutes for an applet to load for the first time. This wait time is restricted to the first time any of the applets are loaded during a WWW session; once the applet is loaded, it is cached in memory, so that only the parameters specific to a particular Workbook exercise (such as the image file and the polygon coordinates) are loaded by the computer for subsequent questions.

Java can use several types of window layouts; the method used for the Workbook applets is called GridBagLayout. This layout allows for customized placement of different display elements, such as the canvas where the geologic feature is displayed, buttons, and text fields. Figures 3-1 and 3-2 are templates for the standard horizontal and vertical layouts in a Name and Outline Workbook exercise. The layout consists of two main panels; one panel is used to display the image of the geologic feature, and the other panel is used to display the buttons and text fields. The button-and-text panel, in turn, is comprised of smaller panels, including a panel for naming-specific fields, and a panel for outline-specific fields. The Name and Outline applet uses both button-and-text panels, while the Name Only and Outline Only applets use only the naming panel or the outlining panel. Because pictures may have a portrait\(^1\) or landscape\(^2\) orientation, there are two standard layouts for the Workbook files. The horizontal layout is used for landscape-oriented pictures, and the vertical layout is used for portrait-oriented pictures.

The applets read several different parameters from the HTML file which calls the applet. The geologic feature name, the image filename, the dimensions of the image, and the coordinates for the bandwidth polygon are specified in HTML files and read into the applet at run time. Sequential data (like the bandwidth polygon coordinates) are parsed by the

---

\(^1\)The height dimension is larger than the width dimension.

\(^2\)The width dimension is larger than the height dimension.
Figure 3-1: Template for a Horizontal Layout Workbook Page
Figure 3-2: Template for a Vertical Layout Workbook Page
applet and stored as individual values in an array. These data are then used by the applet for the specific Workbook question they pertain to; when a new HTML page is accessed, new data is passed to the applet.

3.3.1 Naming Classes

For Workbook sections where students need to provide the name of a given geologic feature (the Name Only exercises and the Naming section of the Name and Outline exercises), the student’s answer is compared with the correct solution string. The correct answer is supplied to the applet via the HTML page, and the student’s answer is compared with that string. If the strings match, the student is told his answer is correct. If the strings do not match, the student is informed that his answer is wrong.

After the student has submitted an answer, he can display the correct answer by clicking the Show Answer button; the correct answer string is displayed in the text field below the student’s answer. The student may also clear any typed or displayed answers by selecting the Clear Text button; text in any of the text display boxes is erased when the button is clicked.

3.3.2 Outlining Classes

For Workbook sections where students need to outline a geologic feature (the Outline Only exercises and the Outlining section of the Name and Outline exercises), the outlines are checked for correctness with two methods.

To create a polygon, the student must click vertices of a polygon with the mouse around the geologic feature, and she must close the outline before she will be able to check her answer. The applet compares the coordinates of each new vertex that the student clicks with the coordinates of the first vertex that was selected. If the new vertex is within eight pixels\(^3\) in the \(x\)- and \(y\)-directions, the applet snaps the new vertex to the initial vertex in order to close the polygon [Kinnicutt, 1991].

Each image used in Workbook exercises which require outlining has two reference polygons associated with it: an outer polygon, which is drawn completely outside of the feature

\(^3\)Picture Element. A pixel is a square grid element whose dimensions are one grid unit by one grid unit.
to be identified, and an inner polygon, which is drawn near the center of the geologic feature. These two polygons form a *bandwidth polygon* which defines the loci of possible correct student vertices [Noack, 1988].

Once the student's closed polygon has been submitted for checking, the applet first calculates the areas of the two reference polygons and the student's polygon. The algorithms described by Noack [Noack, 1988] are used to check for correctness; the actual coding is derived from Kinnicutt's C code [Kinnicutt, 1991]. The standard polygon-area function

\[
\frac{1}{2} \sum_{i=0}^{N-1} (x_{i+1}y_i - x_iy_{i+1})
\]

is used for the area calculations. If the student's polygon has an area greater than the outer reference polygon or is smaller than the inner polygon, then the student's answer is deemed incorrect, no further checking is done, and the student is told that his answer is incorrect. These checks are executed to ensure that the student does not draw a polygon that encompasses the entirety of the displayed picture, or that only covers a tiny amount of the geologic feature. If the student's polygon has an area that is numerically between the areas of the reference polygons, then a second check is executed. Each of the vertices of the student's polygon is checked to determine if it falls within the bandwidth polygon. A built-in Java method for determining if a point is inside a polygon is used for this check. If the point falls within the bandwidth polygon, it is considered correct; otherwise, it is considered incorrect. The number of correct and incorrect points are totaled, and the percentage of incorrect points is calculated. The student's polygon is considered correct if fewer than 40% of the vertices are located outside the bandwidth polygon.

The student can display the correct solution once she has submitted an answer; the bandwidth polygon is displayed on the screen along with the student's polygon when the Show Outline button is selected. The student can also clear all outlines from the picture by selecting the Clear Outline button. Variables are reset to their default states, and image of the geologic feature is redrawn, erasing the polygon outlines.
Chapter 4

Nomad/KRIBS

4.1 Introduction

Nomad is a subsurface profiling program that was developed at M.I.T. It creates simple cross sections from borehole data, as well as kriging the data to produce probabilistic subsurface profiles.

This chapter starts with a short review of the previous work that has been done with Nomad over the past decade. A brief overview of Nomad’s functions is then included. The chapter then describes the variogram modeling algorithms used in the UNCERT and GSLIB geostatistical packages. A description of the application of the UNCERT algorithms to Nomad follows. Finally, the chapter closes with recommendations for further work to be done with Nomad/KRIBS.

4.2 Nomad/KRIBS Description

4.2.1 Nomad/KRIBS History

Like the Physical Geology Tutor (Ch. 2), Nomad has existed in several different formats over the past ten years. Noack wrote the original Nomad as the second module of the Engineering Geology Educator [Noack, 1988]. Noack’s Nomad was written in C and was a ground profiling program which allowed students to draw cross-sections of sites based on borehole data.
It made use of the GEPSE shell\(^1\) to link an expert system that provided feedback about the profile currently being drawn. Noack's Nomad was written using X10 libraries, and when Athena upgraded to X11, Nomad required rewriting. Patrick Kinnicutt [Kinnicutt, 1991] rewrote Nomad for the X11 libraries and the Motif Windowing System; Kinnicutt's Nomad originally did not include the use of an expert system. The original X11 version of Nomad also only ran as a standalone program on monochrome Athena workstations. Kinnicutt later rewrote Nomad in C++, made use of color for X11 systems, and made significant additions to the program's modeling capabilities [Kinnicutt, 1995]. Kinnicutt implemented the KRIBS algorithm, which combined Indicator CoKriging with Bayesian updating techniques in order to model the subsurface; a Decision Navigator was also written so that users could enter subjective data, document the thought process, and easily keep track of multiple modeling paths [Kinnicutt, 1995]. Kinnicutt recently upgraded the kriging module [Kinnicutt, 1998] with an Indicator Kriging algorithm from the GSLIB geostatistical library [Deutsch and Journel, 1997]. This algorithm solves the entire kriging system, rather than only the selected orientation as was previously done.

4.2.2 Nomad/KRIBS Program Features

Nomad displays borehole data three-dimensionally in a block diagram; three orthogonal views are displayed simultaneously on the screen. Figure 4-1 shows an example Nomad project. Nomad can translate and rotate the data block, as well as zoom in and out. Different labels describing ground types can be displayed, and the colors associated with particular ground types are customizable. Boreholes can be added manually, and lines and curves connecting ground boundaries or boreholes can be drawn.

The KRIBS model determines the probability of a given ground type being present at a certain location by using Indicator CoKriging. The data are gridded, and experimental and model semi-variograms are calculated. Both sets of semi-variograms are displayed, and the parameters of the model semi-variograms can then be changed by the user. The variograms are shown in Figure 4-2. Once the user is satisfied with the semi-variogram models, the kriging system can be solved, and the user can then specify a profile to view. Profiles that

\(^1\)The same shell that was originally used for the Physical Geology Tutor
Figure 4-1: Nomad Main Window with Example Data File
Figure 4-2: Experimental and Exponential Model Variograms
display all the ground types can be shown, and the certainty of a ground type being at a specific location can be specified, so that the user can see which ground types are most likely to be present in the profile. A ground profile is shown in Figure 4-3 Contour diagrams of a single ground type can also be displayed for a profile, showing the likelihood as a percentage of a particular ground type being in a given location.

4.3 Semi-Variogram Modeling

4.3.1 The Semi-Variogram

Semi-variograms\(^2\) are used in geostatistics to determine the variability between pairs of data points. Between any two points there exists a distance vector \(\mathbf{h}\). As the separation distance between any given point and the surrounding points increases, the values at each data point tend to become less similar. When the data value of a point is plotted against the value of another point, the resulting data points scatter around the line \(y = x\). Pairs of points with similar data values will plot closer to the \(y = x\) line, and pairs with more divergent data values will plot farther away from the line [Isaaks and Srivastava, 1989].

\(^2\)The term variogram is also used interchangeably to refer to the semi-variogram.
Figure 4-3: Ground Profile Obtained by Using KRIBS
The semi-variogram, which is the name given to the relationship between the moment of inertia about the $y = x$ line and $h$, is determined by taking half of the average squared difference between the values at two data points. The equation for the semi-variogram is as follows:

$$\gamma(h) = \frac{1}{2N(h)} \sum_{(i,j)|h_{i,j}=h} (v_i - v_j)^2$$

The semi-variogram values are plotted against the magnitude of $h$ to obtain a semi-variogram plot [Isaaks and Srivastava, 1989].

As the magnitude of $h$ increases, the variogram values tend to reach a plateau, called the sill. The distance at which this plateau is reached is called the range. Sometimes there is a jump in the data values for distances very close to zero; at zero distance, the value of the variogram is zero, but for small distances away from zero, the variogram can be significantly greater than zero. A semi-variogram value can be extrapolated back to the $y$-axis; this value is called the nugget value.

Semi-variograms are used in kriging to determine the influence that a known data value at a point has on estimating an unknown data value at a different point.

### 4.3.2 GSLIB and UNCERT Packages

UNCERT is a geostatistical calculation package developed at and available from the Colorado School of Mines [Wingle et al., 1997]. UNCERT, in turn, is based on the algorithms available from the GSLIB geostatistical package of Deutsch and Journel [Deutsch and Journel, 1992, Deutsch and Journel, 1997]. The GSLIB package is coded in FORTRAN, and has a text-only interface. The UNCERT programs are coded in C, and have a graphical user interface that operates on a number of platforms, including Unix systems running X11. Both packages contain algorithms for calculating semi-variograms, kriging functions, and other geostatistical functions. A more extensive description of the algorithms used in the UNCERT and GSLIB packages can be seen in the respective user’s manuals [Wingle et al., 1997, Deutsch and Journel, 1997].
4.3.3 KRIBS Implementations

The algorithms used in UNCERT are well-tested and well-documented. To add more functionality to Nomad/KRIBS, several variogram-related algorithms from UNCERT have been incorporated into the Nomad/KRIBS code. Specifically, the 2D variogram surface modeling algorithms and the nested variogram model algorithm are used in Nomad/KRIBS.

2D Semi-Variogram Surfaces

The 2D variogram surface function calculates a series of 18 directional experimental variograms. The variograms are then plotted, either with all 18 variograms displayed on one set of x-y axes, or as a contour map, which makes the variability of the variograms more apparent.

The user specifies the lag distance for the data set; the lag distance is the base separation distance used, and each consecutive h value is a multiple of this distance. The lag tolerance is chosen to be one-half of the lag distance; the lag tolerance is the amount of distance before and after the lag distance where points will be used in calculating the semi-variogram value for that particular lag. A five-degree half angle is used, and the user is able to specify a bandwidth distance, which is also used for choosing points to use in the experimental determination. The half-angle specifies radially how far away from the variogram principal direction that points may be chosen, and the bandwidth value is applied after the first lag distance so as to minimize the effect of points which are farther away. Figure 4-4 (modified from Wingle [Wingle et al., 1997]) shows the principles of the directional variograms and the different parameters used for calculating the directional semi-variograms.

The semi-variogram values can be displayed in two ways: as a $\gamma(h)$ vs $h$ plot with the values for all 18 calculations displayed on the same set of axes, or as a contour surface. An example of the $\gamma(h)$ vs $h$ plot is shown in Figure 4-5. An example of a contoured two-dimensional variogram surface is shown in Figure 4-6. The origin (0, 0) is at the center of the graph; the x and y directions are the different separations distances. The darker shadings in the center of the graph indicate lower $\gamma(h)$ values, while the contours toward the edges of the graph indicate higher $\gamma(h)$ values. Figures 4-7 and 4-8 show two directional variograms from Figure 4-6. Figure 4-7 runs from the center of the contour surface to the top of the
Figure 4-4: Basic Principles of and Parameters Used for Calculating Directional Variograms

graph, while Figure 4-8 runs from the center of the contour surface to the right edge of the graph. It is important to note that while the 2D variogram surface shows anisotropies in the directional variograms, the 2D variogram itself should not be used for performing kriging operations. The 2D variogram is not a continuous surface; rather, 18 variograms are determined, and values not calculated in these 18 variograms are interpolated. For kriging, it is better to use a semi-variogram model where a value can be calculated for any location than to use a 2D variogram surface [Wingle et al., 1997].

**Nested Variogram Models**

When determining a variogram model to use for kriging systems, it is sometimes desirable to combine variogram models to allow the model variogram to more closely approximate the experimental variogram. The nested variogram function allows up to five variogram models to be specified; these models may be of the same or different types (i.e., two nested spherical models, one spherical, one gaussian, and one exponential model, etc.). The user can specify the horizontal, vertical, and isotropic ranges and sills for each nested model, as well as the
Figure 4-5: 18 Directional Variograms Plotted on Same $\gamma(h)$ vs $h$ axis. The legend of symbols is shown to the right of the graph; each symbol represents a directional variogram.
Figure 4-6: Example Contour Graph of a 2D Variogram Surface (data from Isaaks and Srivastava [Isaaks and Srivastava, 1989])

Figure 4-7: Directional Variogram (Oriented Up-Down)
degree of anisotropy in the $x$-, $y$-, and $z$-directions for each variogram model. The same nugget value is used for all nested models for a given semi-variogram. Variogram models using an exponential model, a spherical model, and a nested spherical and exponential model are shown in Figures 4-2, 4-9, and 4-10. Nested variogram models can be used to better approximate the experimental variograms since different sills, ranges, and anisotropies can be specified for each nested model. Variations exhibited in the experimental variogram that cannot be modeled with a single model may be adequately accounted for when using two or more nested models.

4.4 Recommendations for Future Work

Nomad has been enhanced and rewritten several times over the last ten years, and several authors have made recommendations for future development of Nomad and KRIBS. The following suggestions for future work are meant to augment the changes suggested by Kinnicutt [Kinnicutt, 1995], not to supersede them.

The kriging function currently used by KRIBS is a FORTRAN object from the GSLIB geostatistical library which is linked with Nomad at compilation time. The FORTRAN $f77$ compiler used by Athena computers is written in C, so the object files created by it are readily compatible with C/C++ object files compiled on Athena. Problems may arise if Nomad is compiled by users whose FORTRAN and C++ compilers do not produce
Figure 4-9: Experimental and Spherical Model Variograms

Figure 4-10: Experimental and Nested Exponential and Spherical Model Variograms
compatible types of object files, and modifications to both the GSLIB FORTRAN code and
the Nomad/KRIBS C++ code may be necessary. If for some reason a user does not have
a FORTRAN compiler, the problem of even creating an object file from the kriging code
exists. Ideally, the kriging function should be rewritten in C++ to eliminate any potential
compatibility problems.
Chapter 5

Conclusion

5.1 Contributions

Contributions of this thesis include:

- **Rewriting the Physical Geology Tutor in an HTML/CGI/Java format suitable for using with a graphics-based web browser.** Rewriting involved reformatting the Textbook pages to make them comply with the HTML standard, creating a Perl/CGI form template that checks answers to be used for Quiz questions, and coding three Java applets to emulate the Workbook page behavior present in previous versions of the PGT. The WWW format makes it easier for M.I.T. students to access the Tutor, and it paves the way for easy off-site distribution once copyright issues are resolved.

- **2D Variogram Surface Modeling.** A function new to Nomad/KRIBS was added to let users display a two-dimensional variogram surface. A series of 18 directional semi-variograms are calculated, and the data are plotted as either as 18 semi-variogram plots on a single $\gamma(h)$ vs $h$ set of axes or as a contour surface of the semi-variogram. Two-dimensional semi-variogram surfaces allow users to more readily determine directions of anisotropy.

- **Nested Variogram Modeling.** The algorithms used to calculate model semi-variograms were modified to allow nesting of semi-variogram models. Nested var-
iograms can be used to fit the model variogram more closely to the experimental variogram.

5.2 Recommendations

Several recommendations for future work on the Physical Geology Tutor, Nomad, and KRIBS have been suggested in previous chapters. These recommendations are summarized below.

- **Additional media formats.** Video footage and animated pictures could be added to Textbook pages. These would allow students to see geologic processes in action.

- **New Textbook content.** New chapters, such as mass wasting and an introduction to mineralogy could be written.

- **Virtual field trips.** Pictures from course field trips would allow students to refresh their memories of what features were seen on the trips. Students who did not attend the field trips would also have an opportunity to see the features, and all students would be able to view pictures of features that were not seen on the day of the field trip due to poor weather, foliage, or construction.

- **Regexp matching of Workbook answers.** Regular expression matching in the Workbook section would allow students’ answers to be considered correct if most of the letters in the students’ answers matched the solution string. Regular expression libraries could be used, or multiple correct answer strings could be compared with the students’ answers.

- **Rewrite FORTRAN kriging function into C++.** This would eliminate the necessity of a compatible FORTRAN compiler and would ensure that the kriging function would reliably work when invoked.
Appendix A

HTML Physical Geology Tutor
Administrator’s Manual

A.1 Introduction

The Physical Geology Tutor (PGT) consists of 8 chapters dealing with topics in physical geology. Each chapter is made up of several parts: a Textbook section, a Quiz section, a Workbook section (with the exception of the Lacustrine chapter), and an Index. There is an additional whole-PGT Index which can be modified in a manner similar to that of the chapter Indices.

The structure and modification techniques for these files are described below. A section covering Unix commands and utilities is presented first. This section describes how to change directories and manipulate files on Unix systems; it also provides basic instruction for using the Emacs text editor and the XV image display program. A section on PGT Resources is then presented, which contains information about the location of extra image files on cfd4.mit.edu, obtaining digital images and converting photographs and slides to digital formats, and where to find help about different topics addressed in managing the Physical Geology Tutor. The Textbook section describes how to write a new Textbook file in HTML. The Quiz section tells how to modify a template file to create a new Quiz entry, and the Workbook section describes how to add information to Workbook datafiles, generate new HTML files by using a script program and the datafiles, and regenerate the
executable Java applets from the Java source code. Following the sections dealing with the major PGT components is a section containing information about putting newly-created pages into the Tutor in the correct locations and editing information in existing files so that the Navigation Bars and other navigational features will take the user to the correct files. This manual ends instructions for restarting the program which runs the webserver on cfd4.mit.edu.

A.2 Unix Background

The PGT is currently on the computer cfd4.mit.edu, a Sun SPARCstation4 running Solaris 2.5.1 and Athena release 8.1.x. It is supported by M.I.T.’s Information Systems. The machine is physically located at the time of writing in 1-331.

A.2.1 Accessing the /mit/1.38 and /mit/1.38_dev Project Athena Lockers

The directories /mit/1.38 and /mit/1.38_dev are lockers\(^1\) which are stored and served by Project Athena. Source code, datafiles, and documentation for the Tutor can be found in these directories. To add either of these lockers, type \texttt{add lockername} at the athena\% prompt. For example, to add the 1.38_dev directory, type \texttt{athena\% add 1.38dev}. Now the 1.38_dev locker can be accessed by typing \texttt{athena\% cd /mit/1.38_dev}. If the locker has not been added, neither the locker nor the files contained within it can be accessed by using the /mit/lockerno\%me naming scheme.

A.2.2 Starting Netscape

Netscape is a program used to display HTML pages from the World Wide Web (WWW). Netscape 2.x and all subsequent releases of the program are enabled to display Java applets in the browser window. Any web-browser that is Java-compatible can be used to view the PGT; Netscape was used in the development of the HTML-based Tutor because of its availability on Athena and its ease of use.

\(^1\)Filesystems which may be mounted for use on an individual Athena workstation.
Netscape can be started on an Athena workstation (Sun and SGI) from the DASH menubar. Select the Communication item from the menubar, and choose the World Wide Web arrow from the menu that pops up. From the World Wide Web menu, choose the Netscape arrow, and from the Netscape menu, click the mouse on any of the Netscape menu items (click on the actual menu item and not the question mark). Figure A-1 shows the DASH menus that will appear on the screen. DASH will ask if it should start Netscape; click on the yes button. A Netscape window will be created within several seconds. Delete the text that is found near the top of the page on the "Location:" line and type the URL of the Tutor: http://cfd4.mit.edu/PGTprivate/ and press the Return key on the keyboard. The Title Page for the Physical Geology Tutor will appear in the browser window (Figure A-2).

Netscape can also be started from an xterm window. At the athena% prompt, type
add infoagents; netscape http://cfd4.mit.edu/PGTprivate/ &
The Physical Geology Tutor Title Page will appear in the browser window.

A.2.3 Copying Files in Unix

Several template files have been created to make the addition of new pages a simpler process. Copies of the template files should be made when creating a new entry for the Tutor so that the original template file is not changed or deleted.

On Unix systems, the syntax used to make a copy of a file with a different filename is cp existingfilename newfilename. For example, the command to copy the original file template.html to a new file volcano.html is athena% cp template.html volcano.html.
Physical Geology Tutor

HTML Version 1.0
Department of Civil and Environmental Engineering
Massachusetts Institute of Technology

Note: You may need to resize your browser window to see the entire page properly.

- Using the Physical Geology Tutor
- Foliar Chapter
- Fluvial Chapter
- Glacial Chapter
- Karst Chapter
- Lacustrine Chapter
- Shoreline Chapter
- Structural Chapter
- Vulcanism Chapter
- PGT Main Index
- Photograph and Illustration References
- Questions, Comments, Ideas
- PGT Survey

Figure A-2: The Physical Geology Tutor Title Page
A.2.4 Using Emacs

Emacs is a text editor on Unix systems. It (or another text editor) will need to be used to edit new or existing files for the Tutor. Figure A-3 shows the main Emacs screen. To edit a file with Emacs, type `emacs filename` at the `athena%`. For example, to edit a file called `volcano.html`, type `athena% emacs volcano.html`. Once the Emacs window has appeared on the screen, the arrow keys may be used to move the cursor about the document, the delete key may be used to delete characters, and the keyboard can be used to make additions or changes to the text. To save the file, hold down the Control key and then press the `x` key (C-x), let go of both keys, and then press Control-s (C-s). When editing is complete, quit Emacs by pressing Control-x (C-x) and then Control-c (C-c). If the file has been changed since the last save, Emacs will ask if the current version of the file should be saved; type y for the answer.

A.2.5 Using XV

XV is an image display program for Unix computers running the X Window System. It can be used to view images, perform such editing tasks as cropping, enlarging, and reducing pictures, and in a roundabout way, finding the coordinates for polygons used in the PGT's Workbook section.
Starting XV

XV can be started from the DASH menubar at the top of the screen in an Athena session. Click the left mouse button once on the Text/Graphics item of the DASH menu. A new menu will open below the words Text/Graphics; click on the word Graphics. A new menu will appear to the right of Graphics; click once on the word XV that appears in the new menu. Figure A-4 shows the DASH menus used to start XV. A dialogue box will appear asking if XV should be started. Click the mouse on the Yes button. An outline of a window will appear on the screen; click the left mouse button to make the outline into the XV window. The XV window will say XV by John Bradley (Figure A-5). Click the Right mouse button on the XV window to open the xv controls window.

Figure A-5: The Initial XV Window
Viewing and Manipulating Pictures

The main menu window for XV is shown in Figure A-6. This window is called **XV controls**. To load an image into XV, click the left mouse button on the **Load** button on the right side of the screen. Another window will pop up, listing all the files in the directory in which XV was started. Click once on the filename of the file to be opened, and then click the **OK** button. The selected image will replace the XV by John Bradley image. Other pictures can be loaded in a similar manner, but only one image can be displayed at a time. To change between pictures that have already been loaded, double click on the desired image name in the **XV controls** window, and the newly-selected picture will replace the previous picture on the screen.

![Figure A-6: The XV Menu Window](image)

Pictures can be reduced in size in XV; this function is particularly useful for creating the “thumbnail” (very small) images used on the PGT Textbook pages. Once a picture is displayed, put the pointer on XV window showing the picture and type a capital **S** on the keyboard. A **Prompt Window** will appear, asking for a new image display size. Thumbnail pictures are either 100x75 pixels (for pictures which are wider than they are tall) or 76x100 pixels (for pictures which are taller than they are wide). Delete the default size that is displayed in the Prompt Window, enter the appropriate thumbnail dimensions,
and click once on the Ok button. The image will be resized to the new dimensions. While the pointer is still on an XV window, type a small s on the keyboard to smooth the colors of the image.

After creating a small image, it is necessary to save it. To do this, click once on the Save button on the xv controls window. The xv save window will pop up, prompting for a name (Figure A-7). The current naming convention for the thumbnail pictures is the name of the large picture with the word “tiny” appended to it. For example, a large image called hheffjord.jpg would have a thumbnail image called hheffjordtiny.jpg. Photographs in the PGT are saved as JPEG\(^2\) files; the files should have the extension “.jpg” at the end of the filename. A button next to the word “Format:” appears at the top of the xv save menu. Press and hold down the left mouse button while a new menu appears. Keeping the mouse button depressed, move the mouse down until the word JPEG is highlighted. Release the mouse button to accept that choice. When the appropriate filename has been entered and the Format type has been set to JPEG, click the mouse button on the Ok button on

![Figure A-7: The XV Save Window](image)

2Joint Photographic Experts Group. This is the name of the original committee that wrote the JPEG standardized image compression algorithm. The image format that uses the JPEG algorithm is called JFIF, but the name JPEG is often used to refer to images stored in the JFIF format.
the \texttt{xv} \texttt{save} screen to save the image.

**Finding Coordinates for Workbook Polygons**

XV can be used to obtain the coordinates for the bandwidth polygon used in the Workbook section of the Tutor. This method is not particularly efficient, but satisfactory results can be obtained.

GIF$^3$ and JPEG images are drawn by assigning different colors to squares on a grid. The origin of the grid is at the top left corner of the image, and that square (pixel) is assigned the coordinates $(0, 0)$ (Figure A-8). The $x$-axis extends to the right, and the $y$-axis extends toward the bottom of the picture (the $y$-values are positive in this direction). XV will show the coordinates of the pointer position if the \texttt{middle} mouse button is depressed and held down. A bar will appear at the top or bottom of the image with a series of numbers in it. The two numbers at the left side of this bar are the $x$ and $y$ coordinates. If the mouse is dragged (moved while a button is depressed) over the picture, the coordinates for different locations on the picture can be obtained. Figure A-9 shows how the coordinates are displayed on the

\begin{figure}[h]
\centering
\includegraphics[width=0.4\textwidth]{image_coordinate_system}
\caption{Image Coordinate System}
\end{figure}

\footnotetext[3]{Graphics Interchange Format}
Figure A-9: XV Image Showing Coordinates of Mouse Pointer

screen. Polygon vertices can be obtained by dragging the mouse and sketching an invisible polygon around a geologic feature. The bar that displays the coordinates will disappear when the middle mouse button is released. There will be no lines drawn while the mouse is dragged, and the coordinates will need to be written down by hand to be later entered in the appropriate Workbook datafiles.

A.3 PGT Resources

This section describes the locations of images used in the PGT and methods for obtaining new digitized images. It also suggests where to get help with matters related to the PGT.

A.3.1 Image Locations

There are many digitized images used in the PGT, as well as a number of digitized images that are not currently being used. Pictures used in the chapter Textbooks and Workbooks are stored within the individual chapter directories in a subdirectory called pictures/. For example, pictures used in the Glacial Textbook and Workbook are stored in /var/home/etc/httpd/htdocs/PGTprivate/glacial/pictures/. Pictures from
Jennie Hango's photo collection which are not currently used in the Tutor can be found in the directory `/var/home/pgt/jshslides/`. An annotated index of the images is provided.

**A.3.2 Obtaining Digitized Images**

Digitized images can be obtained in a number of ways. Images that are on the Photo CD-ROMs (most of the photographs used in the Textbooks and the Workbooks) can be viewed on Macintoshes or PCs using the viewing software included on the CD-ROMs. A selected image can be saved onto the computer's hard drive in JPEG format, and then the file can be transferred to Athena via Fetch (Macintosh) or FTP (PC).

Photographs and drawings that are not on a CD-ROM or are not already in a digital format on a computer will need to be scanned. The Media Station in Academic Computing Services has a flatbed scanner which can be used to scan photographs and drawings. Contact the Information Services Faculty Liaisons for more information. Images on 35mm slides need to be scanned using a special slide scanner. The Computer Resource Lab (CRL) in the Department of Urban Studies and Planning has a slide scanner that may be available for use; contact the CRL lab managers at crlhelp@mit.edu for more information and to obtain permission to use the lab.

**A.3.3 pgt-admin@mit.edu**

An e-mail alias has been obtained for the PGT. The address pgt-admin@mit.edu will forward e-mail to the PGT Administrator, as well as to any others whom the Administrator has added to the pgt-admin mailing list. There should always be at least one person receiving mail sent to pgt-admin. The Administrator can add people to the mailing list by using the Athena listmaint utility's option #6. Additional Administrators for the mailing list can be added by sending mail to accounts@mit.edu with the name of the list (pgt-admin) and the names and e-mail addresses of the people who should be able to add or remove names from the mailing list.
A.3.4 Finding Help

Athena’s On-Line Consulting (OLC) service is useful for questions about using Athena. Their Hotline number is x3-4435, and they may be contacted on-line by typing olc at the athena% prompt and following the directions on the screen. An Athena mailing list called java-hackers@mit.edu is used as a forum for all topics Java. The people on the list can help with questions pertaining to Java. The Information Systems Faculty Liaisons can answer questions about courseware development and locker quotas. Their e-mail address is f.l@mit.edu. Jennie Hango can be reached at jhango@alumni.hmc.edu; ask her anything about the PGT. While she probably won’t have access to Athena after graduation, she can troubleshoot via e-mail.

A.4 Textbook

A.4.1 HTML Files

An HTML file is essentially a text file with additional formatting or “mark-up” tags. Tags are formatting codes that tell the web browser how to display text and graphics elements in the browser window. Any text that starts with a “less-than” sign (<) and ends with a “greater-than” sign (>) is an HTML tag. The formatting tags are interpreted by the web browser and are not displayed on the screen. Most HTML mark-up tags have the style of <tag>marked-up text</tag>. A minimal HTML file resembles the code shown in Figure A-10.

Most tags consist of two parts: an opening tag (<tag>) and a closing tag(</tag>). The opening tag is placed before the text that is to be modified, and the closing tag is placed immediately following it. The closing tag is similar in appearance to the opening tag, with the addition of a slash (/) immediately following the less-than (<) sign.

Tags can be nested, i.e., two sets of nested tags should look like this:

```html
<strong><a href="mailto:jruser@mit.edu">jruser@mit.edu</a></strong>
```

In this case, the <a href=""/> </a> tag was nested within the <strong> </strong> tag.

A few tags do not require a closing tag. These tags include: <p> (end of paragraph),

---

[^4]: HyperText Markup Language

70
An important tag in HTML is the anchor or anchor link tag. The anchor tag is used to hyperlink a word, phrase, or image in an HTML document to another HTML document. Hyperlinks tell the web browser to go to a different page when the user clicks the mouse button on the hyperlinked word. A hyperlinked word is displayed in blue, underlined text in most graphics-based web browsers, and as underlined or inverse text in text-based web browsers. An anchor tag resembles the following line:

```
<a href="http://cfd4.mit.edu/PGTprivate/">Physical Geology Tutor</a>
```

This tag has the effect of hyperlinking the words “Physical Geology Tutor.” When the mouse button is clicked on the term, the web browser will go to the webpage that is at http://cfd4.mit.edu/PGTprivate/, which in this case is the Title Page of the Physical Geology Tutor. The `a` in the `<a>` and `</a>` tags stands for “anchor”. The `href="..."` is the hyperlink reference, or the location of the file that the browser should access when
the hyperlink is selected with the mouse. The data required here are URLs\textsuperscript{5}, often in the form \url{http://cfd4.mit.edu/PGTprivate/} or \url{mailto:pgt-admin@mit.edu}. The URLs should be placed within the quotation marks that follow href.

### A.4.2 Editing and Adding Textbook Files

The directories on the \texttt{cfd4.mit.edu} hard drive where the Textbook HTML files are stored are listed in Figure A-11.

```plaintext
/var/home/etc/httpd/htdocs/PGTprivate/eolian/
/var/home/etc/httpd/htdocs/PGTprivate/fluvial/
/var/home/etc/httpd/htdocs/PGTprivate/glacial/
/var/home/etc/httpd/htdocs/PGTprivate/karst/
/var/home/etc/httpd/htdocs/PGTprivate/lacustrine/
/var/home/etc/httpd/htdocs/PGTprivate/shoreline/
/var/home/etc/httpd/htdocs/PGTprivate/structural/
/var/home/etc/httpd/htdocs/PGTprivate/vulcanism/
```

Figure A-11: Location of PGT Textbook Files

A template file for the Textbook pages is found in the \texttt{/var/home/etc/httpd/htdocs/PGTprivate/templates/} directory. This file is called \texttt{template.html}. \textbf{Do not edit this file!} Make a copy of this file and give the copy the name of the new Textbook page which is being created. All HTML Textbook files should use a basic skeleton that resembles the code given in Figure A-12. An example Textbook file can be found in Appendix C, and the standard template file is in Appendix D.

The words “Geologic Term” in the third line of code should be replaced with the name of the chapter and feature being discussed. The series of tags immediately following the \texttt{<body>} tag comprise the code for the Navigation Bar for the Tutor, as shown in Figure A-13. The \texttt{prevpage.html} filename should be replaced with the filename of the previous page, and likewise the \texttt{nextpage.html}, and \texttt{chaindex.html} filenames should be replaced by the appropriate filenames. An identical Navigation Bar is found near the bottom of the page below the images, so care should be taken to make sure the two Navigation Bars are identical. The text description of the term should be placed between the \texttt{<font>} and \texttt{</font>} tags.

\textsuperscript{5}Uniform Resource Locators. “Addresses” for data on the WWW.
If a new Textbook entry uses more than one picture, a different page will need to be created for each picture. The naming scheme is as follows:

`textbook.html  textbook2.html  textbook3.html`

The only things that should differ among the pages are the full-sized images displayed and the corresponding annotations and references.

### A.5 Quiz

Quiz files are CGI\(^6\) files written in the Perl programming language. The directories where the files are stored in `cfd4.mit.edu` are listed in Figure A-14. Templates for the Quiz script pages can be found in `/var/home/etc/httpd/htdocs/PGTprivate/templates/` under the filenames `quizsingle.cgi` and `quizmultiple.cgi`. These files are also listed in Appendix D, and

---

\(^6\)Common Gateway Interface
example Quiz files are listed in Appendix C. The first file is for Quiz questions with a single
correct answer, and the second file is used when there is more than one correct answer for
a Quiz question. **Do not edit these files**; make copies of them and give the new files
the names of the new Quiz pages being created. For example, to create a new Eolian Quiz
question which will be question #87, type `athena% cp quizsingle.cgi eolquiz87.cgi`. Quiz files need to be made executable so that they can be run when someone accesses the
Quiz section, so, continuing with the example file, the command `chmod 755 eolquiz87.cgi`
would need to be typed at the athena% prompt. The file is now readable, writable, and
executable by the file’s owner, and readable and executable by everyone else. To edit the
new Quiz file, type `athena% emacs eolquiz87.cgi`.

The first section of code that needs to be modified is located in the **MAIN** section
of the program, shown in Figure A-15. The two filenames that are listed after `$url` and

```bash
### Main

MAIN:
{
    local ($url, $url2);
    $url="eolquiz1.cgi";
    $url2="eolquiz3.cgi";

    if (&ReadParse(*input)) {
        &ProcessForm;
    } else {
        &SubmitForm;
    }
}
```

Figure A-15: The MAIN Function for a Quiz Question

$url2$ need to be changed to the names of the files that come immediately before and
immediately after the current file. For example, a file called `eolquiz144.cgi` would need to
have `$url=eolquiz143.cgi` and `$url2=eolquiz145.cgi`.

The next section of the file that needs to be edited is in the **Question** subroutine, shown
in Figure A-16. The name of the current file is listed next to the word “action”; make sure
that the filename listed there is really the same as the current filename. The Question
## Question --

sub Question
{
    print "<form method="post" action="eolquiz2.cgi">
    <h2>Question #2</h2>
    <font size=+1>
Which describes the orientation of a yardang with respect to the wind?<p>
EOT

  if ($input{'question'} eq "rightl") {
  print "<input type="radio" name="question" value="rightl" checked>parallel<br>
EOT
  } else {
  print "<input type="radio" name="question" value="rightl">parallel<br>
EOT
  }

  if ($input{'question'} eq "wrong2") {
  print "<input type="radio" name="question" value="wrong2" checked>subparallel<br>
EOT
  } else {
  print "<input type="radio" name="question" value="wrong2">subparallel<br>
EOT
  }

  if ($input{'question'} eq "wrong3") {
  print "<input type="radio" name="question" value="wrong3" checked>oblique<p>
EOT
  } else {
  print "<input type="radio" name="question" value="wrong3">oblique<p>
EOT
  }

print "</font>";
}

Figure A-16: The Question Subroutine of a Quiz File
Number is listed in the next line; change it to correspond with the current file as well. The question is listed in the line which follows <font size=+1>. Type in the new question and put a paragraph tag (<p>) at the end of the sentence. It is permissible to have a question take up more than line; the <p> should only be placed at the end of the question. Following the question are a series of if...else statements; the answers are included within these statements. Replace the answers (in Figure A-16 the answers are parallel, subparallel, and oblique). The words “right” and “wrong” are included in the if...else statements; the answer which is correct should have “right” listed in its if...else statement, and the incorrect answers should have “wrong” listed. Do not delete the numbers that follow “right” or “wrong.” Add more if...else statements if there are more answers that need to be listed, and delete if...else statements if there are fewer answers to be displayed.

The last part of the file that needs to be edited is in the ProcessForm subroutine, shown in Figure A-17. The number following the word “right” will need to be changed

```perl
## ProcessForm

sub ProcessForm
{
    &Header;

    if ($input{'question'} eq "right1") {
        &Question;
        &Yes;
    }
    else {
        &Question;
        &SubForm;
        &NopE;
    }
};
&Url;
&Footer;
}
```

Figure A-17: The ProcessForm Subroutine of a Quiz File

to whichever answer is the correct one. For example, if the second answer is the correct answer, the line of code should read
if (($input{'question'} eq "right2") { .

A.6 Workbook

A.6.1 HTML Files and Perl Script

Workbook files are written in HTML. They are generated by a Perl script using datafiles for each Workbook section. The directories which contain the HTML files are listed in Figure A-18. The Workbook data files are located in the /mit/1.38.dev/src/geotutor/wkbkfiles/

/var/home/etc/httpd/htdocs/PGTprivate/eolian/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/fluviial/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/glacial/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/karst/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/lacustrine/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/shoreline/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/structural/workbook/
/var/home/etc/httpd/htdocs/PGTprivate/vulcanism/workbook/

Figure A-18: Locations of the Workbook HTML Files on cfd4.mit.edu
directory.

The Perl script used to generate the Workbook HTML files is called genworkbook.pl and is located in the /var/home/etc/httpd/htdocs/PGTprivate/src/ directory, and is also listed in Appendix F.

The data files follow the naming convention of chaptersectionwb where chapter is the name of the chapter and section is the type of Workbook exercise (one of name, outline, or both). For example, the data file for the eolian Workbook pages that require the student to only type the feature name is called eoliannamewb. The data files are the definitive source for generation of the HTML pages; any changes made to the HTML files will be wiped out when the Perl script is run again. It is possible to edit individual Workbook HTML files, but editing them by hand is not recommended. Since the HTML files can be regenerated by running the genworkbook.pl script, any changes to the individual HTML files will be lost when the script is executed and new (replacement) pages are written. The individual HTML files should not be modified! Instead, make the changes in the appropriate
data file(s) and then rerun the `genworkbook.pl` script to create Workbook HTML pages with the correct information.

The datafiles for the different Workbook sections have similar formats; Figure A-19 shows an example of the data format for an individual Workbook page entry. The first five lines of the file contain data used in each HTML file generated by the Perl script. The first line is the chapter that the Workbook pages will cover. The second line describes the type of Workbook exercise the data file is for. The third line indicates which Java class file to use. The fourth and fifth lines indicate respectively how much extra horizontal and vertical space should be allotted to the applet beyond the size of the displayed picture. These first five lines of information should only appear once in a datafile, at the very beginning of the file. The blank line following the fifth line serves to delimit the initialization data from the individual page information.

The data following the initialization lines are used to generate unique HTML pages; each section of data has 10 lines. Line 7 is the name of the picture to use on the Workbook page. Lines 8 and 9 indicate the width and height, respectively, of the picture. The 10th line is the name of the feature, and the 11th line is the name of the photographic source. The 12th line is a web page associated with the picture. The 13th through 16th lines are

Figure A-19: An Example of the Workbook Datafiles

```plaintext
(1) eolian
(2) both
(3) both.class
(4) 230
(5) 200
(6)
(7) LW1G_1.11.jpg
(8) 486
(9) 365
(10) coppice dune
(11) Ward
(12) fore.html
(13) 265,232,208,235,294,324,330,308,265
(14) 180,192,214,275,248,216,185,180
(15) 195,156,105,73,49,83,136,201,390,431,424,409,379,293,247,195
(16) 101,131,176,225,268,307,337,336,320,280,210,170,120,97,93,101
```
the coordinates for the bandwidth polygon used to check a student-drawn polygon and display a correct outline around a feature. The 13th line contains the x-values for the inner part of the bandwidth polygon, and the 14th line contains the y-values. Similarly, the 15th line contains the x-values for the outer part of the bandwidth polygon, and the 16th line contains the y-values. Variations on lines 7-16 should be repeated in the document for each Workbook item. The individual Workbook page data sets are delimited by blank lines, so a blank line should be placed after each Workbook item’s data set.

To generate the Workbook files for each chapter, change directories to the workbook directory in the appropriate chapter. For example, to change to the workbook directory in the karst chapter, type `cd /var/home/etc/httpd/htdocs/PGTprivate/karst/workbook` at the `athena%` prompt. Run the perl script `genworkbook.pl` from the `athena%` prompt by typing `/mit/1.38.dev/src/geotutor/perlstuff/genworkbook.pl`. The script will prompt for the name of a data file; enter the appropriate data file name (usually resembling `/mit/1.38.dev/src/geotutor/wkbkfiles/eolianbothwb`). The script will generate the files, and it will display on the screen how many files were generated. Look at the contents of the files to make sure that the script worked correctly; if the HTML files do not look right, check the datafile for errors and run the script again. For example, if there are numbers listed in the “featurename” value, such as `<param name=featurename value="187,187,268,267,187">`, there might be a line which has been omitted from an entry in the datafile. Errors in the datafile should be corrected before running the script again. An example of a Workbook HTML file generated with the perl script is shown in Figure A-20. The script will need to be run 3 times for each chapter to generate the HTML files for the three types of Workbook exercises: once for the `chapterbothwb` file, once for the `chapternamewb` file, and once for the `chapteroutwb` file.

A.6.2 Java Source Code and Class Files

The Workbook section uses six Java Classes, which are located on cfd4.mit.edu in the directory `/var/home/etc/httpd/htdocs/PGTprivate/java/`. The class files are the executable program files that the web browsers use to display the Java applet. The Workbook pages need the class files in order to run the Java applet. These class files were generated by
Figure A-20: An Example of a Workbook HTML File Generated by the Perl Script
three Java source files, also located in /var/home/etc/httpd/htdocs/PGTprivate/java/. If the class files are accidentally deleted or corrupted, they will need to be regenerated by recompiling the Java source files. To regenerate the class files, change to the directory where the class files should be located, add the java locker on Athena, and recompile the Java source code. The required commands to do this are listed below.

```
@ena% cd /var/home/etc/httpd/htdocs/PGTprivate/java
@ena% add java
@ena% javac /var/home/etc/httpd/htdocs/PGTprivate/java/Name.java
```

### A.7 Index

The Index files are HTML files located in the same directory as the Textbook files. Their naming follows the convention of `chaindex.html` where `cha` is the first three letters of the chapter name. The Index for the whole PGT is called `mainindex.html` and is located in /var/home/etc/httpd/htdocs/PGTprivate/.

A typical entry from a Chapter Index file looks like this:

```
<dt><a href="blowout.html" >Blowouts</a>
```

The `<dt>` at the beginning of the line is an HTML tag which tells the web browser to display this entry as part of a list. The `<a href="">...<a>` is the link tag (Section A.4.1) which tells the web browser the name of the page to link to and the location of the linked page. The name of the page (`blowout.html` in this case) is specified between the quotation marks (""), and the feature name (`Blowouts` here) should be placed between the `<a...>` and `</a>` tags. To add new entries to the chapter Index files, locate the alphabetic position of the new entry, and add a line that looks like the above example, replacing `blowout.html` with the new page location, and `Blowouts` with the new feature name.

The `mainindex.html` file has a slightly different format than the chapter Index files. A typical line from `mainindex.html` looks like this:

```
<a href="glacial/cirque.html">Basin (glacial)</a>
```

It is important to include the name of chapter for a term as well as a filename.
A.8 Bringing It All Together

Because of the way that the PGT pages are accessed, it is often necessary to update the Navigation Bars at the tops and bottoms of newly added pages, as well as the pages which immediately precede and follow those pages. The pages which may need editing include: pages immediately preceding and following a new Textbook entry, chapter Table of Contents pages, the former last entry of the Quiz questions, the Quiz questions listing pages, the Workbook section pages (eolexboth.html, fluexname.html, glaexout.html, etc.), and the chapter Index pages. The data files which the Workbook page-generating script uses do not need further alteration after new Workbook data have been added; the script automatically puts the correct links into the HTML files. The correct information for the previous and next pages are written directly to each HTML file the script generates. A summary of the updating steps is presented in Figure A-21. An example of the code used for the Navigation Bar is shown in Figure A-13. The filenames prev.html and next.html listed in the example code will need to be replaced with the correct next and previous pages. In most cases, these filenames will end in .html, but if a page immediately follows a Quiz page, the previous page name will end in .cgi. Changing the previous and next pages on a Quiz page is somewhat different. In Quiz files, there are two variables called $url and $url2. $url is the name of the previous file, and $url2 is the name of the next file. The code resembles the following example:

```bash
$url="chaquizX-1.cgi";
$url2="chaquizX+1.cgi";
```

The current file's number is X, and the previous and following files are X-1 and X+1, respectively. The old filename(s) simply needs to be replaced with the new one(s).

Navigation Bars should be edited last when adding new pages. To begin the link updating process, decide where a new file should be located in the PGT; i.e., when a student is using the Tutor, where in relation to other files should she find the newly-added one? If the new file is a Textbook file, decide where it should go in the Table of Contents. If it is a Quiz file, give it the next sequential number that is not used in the appropriate chapter Quiz. For a new Workbook page, determine which Workbook section would be appropriate.
New Textbook Pages

* Create New Page in Directory
* Make File World-Readable
* Decide Where Page Should Go in Table of Contents
* Add Link for Terms on Page to Chapter Index and PGT Main Index
* Edit Navigation Bar on New Page
* Edit Navigation Bars on Pages that Precede and Follow New Page

New Quiz Pages

* Create Quiz File
* Make File World-Readable
* Determine $X$, the Number of Existing Quiz Files, and Name New File "chaquizX.cgi"
* Change the $url$ and $url2$ Variables in .cgi File
* Add Link to Quiz Index Page
* Change Navigation Bar on New Quiz Page
* Change Navigation Bars on Previous and Following Pages

New Workbook Pages

* Add New Information to Appropriate Workbook Datafile
* Run Workbook Page Generation Script
* Update Links on Workbook Header Pages

Figure A-21: Summary of Operations Needed to Link PGT Pages Together
Once a location has been determined, edit the files that have "named links" to this file, i.e., files which refer to the new file by name, rather than just a link that says Next or Prev. For a Textbook file, add a new line to the Table of Contents page where the new file should be listed. A new entry should also be made in both the appropriate Textbook Index, as well as the Main PGT Index. For a Quiz page, edit the Quiz Index page (called chaquiz.html, where cha should be replaced with the first three letters of the current chapter) and add a link to the new page at the bottom of the list of Quiz questions. For a Workbook page, add the new Workbook page's information to the appropriate Workbook data file. After this step has been completed, edit the Navigation Bars on the newly added page, as well as on the pages which come sequentially before it and after it as seen by someone using the PGT. The Next and Prev lines will need to be changed to reflect the new ordering of the pages.

**Do not change the Navigation Bars on the Workbook Pages!** The genworkbook.pl script automatically writes the correct Navigation Bar when the pages are created.

The Table of Contents pages and the Quiz Number Listing pages will need editing if a new Textbook page or Quiz question is added. The Table of Contents will need an entry for the new Textbook page. The entry should be placed as an HTML anchor link in the correct location in the menu. The format should resemble the following: `<a href="newfile.html">x.x.x Feature Name</a>` where newfile.html is the new Textbook page, x.x.x is the chapter and section number for that page, and Feature Name is the name of the feature described.

The Quiz Number listing pages will need to be revised when a new Quiz question is added. The Quiz Number listing pages are HTML files which contain a series of links to the Quiz pages. The entries resemble the following code:

```html
<a href="eolquiz10.cgi">10 </a>  
<a href="eolquiz11.cgi">11 </a>  
<a href="eolquiz12.cgi">12 </a>
```

A new line will need to be created for the Quiz question, following the given format.

When the changes have been made for all the Navigation Bars, check to make sure they work by opening Netscape and navigating through the Tutor using them.
A.9 Restarting the httpd Server

A.9.1 Starting the Server after the Computer Has Been Down

If cfd4.mit.edu has crashed or has been switched off for some reason, the httpd server (the program that lets the website run) will need to be restarted after the computer is switched back on. The Administrator will need to log in as root on the machine. Root is a special account on Unix machines that has access to everything stored on the machine, including files, programs, and currently running jobs. It is sometimes the only account that can run certain programs, like the httpd program. Logging in as or becoming root can be accomplished by typing su (for “SuperUser”) at the athena% prompt. The Administrator will be prompted for the root password, which the Administrator should know, but not write down anywhere. After successfully logging in as Root, the Administrator will need to change to the /var/home/etc/httpd/logs/ directory and type ./start.httpd at the athena% prompt to restart the httpd server. The start.httpd file is a script which contains the following line:

/var/home/etc/httpd/src/httpd -f /var/home/etc/httpd/conf/httpd.conf

The script executes that command and the server begins to run. Log out from the root account by typing exit at the athena% prompt.

A.9.2 Restarting the Server While It Is Currently Running

Sometimes, especially after changing a configuration parameter, it is necessary to restart the httpd server, without having to turn the computer off and back on again. To do this, the Administrator must log in as root and restart the root-owned process that is running the httpd server. Unix machines keep track of which users run which programs; a “process” is a job that a user has started, or a program that the user is running. As long as the program or job is running, a number called the “Process ID Number” is assigned to the program or job. The account that started running the job or program is called the “Process Owner.” There may be more than 10 processes running httpd, but root only owns one of them; this process is the one that needs to be restarted. Log in as root and type athena% ps -ef | grep httpd. This command tells the computer to look for processes
that have the string “httpd” in it. Figure A-22 shows a typical return from the ps command. Look for a listing that has root as its first word:

```
root  551  1  80 Aug 08 ?  0:07
/var/home/etc/httpd/src/httpd -f /var/home/etc/httpd/conf/httpd.conf
```

This listing is the primary httpd process; this is the one that needs to be restarted. To restart the process, first determine its Process ID Number. This is the number immediately to the right of the Process Owner, which is the first item listed for each entry. For the httpd server, the Process Owners are either root or nobody. In this case, the Process ID Number for the root invocation of httpd is 551. The general syntax used to restart a job (i.e., make a job check its configuration files) is `kill -HUP PID`. The “-HUP” tells the computer to restart the process, and PID should be replaced with the process ID number. In the case of our example, the command to restart the httpd server is `athena% kill -HUP 551`. The server will then restart.

A script called `restart.httpd` automates this process. It is located in the `/var/home/etc/httpd/logs/` directory. To restart the httpd server using this script, log in as root and change to that directory. Type `./restart.httpd` at the athena% prompt. The server will then restart.
Appendix B

HTML Physical Geology Tutor
User’s Manual

B.1 Introduction

B.1.1 Getting Started

The Physical Geology Tutor (PGT) is a computer-based courseware package which can be accessed on the World Wide Web via a computer account on Athena, ResNet, or any other M.I.T. network. The Tutor in its current version is a web-based interactive application, and can be accessed from anywhere within the mit.edu domain. Due to copyright issues, the PGT cannot be accessed from domains outside the mit.edu domain.

The URL for the PGT is http://cfd4.mit.edu/PGTprivate/. The PGT is best viewed from the Netscape Navigator web browser, and Java must be enabled within the browser. To run Netscape on Athena, type add netscape; netscape & at the athena% prompt. If Java is not enabled, it will need to be enabled to utilize the entire Tutor. Click on the Options menu item at the top of the window, and choose the Network preferences... item. A menu box will appear; choose Languages from the options shown, and check the Enable Java box. Click on the OK button, and Java will be enabled in this and future Netscape sessions.

Accessing the above URL brings up the Title Page for the PGT (Figure B-1). On this page, the different chapters of the PGT are listed, and the user may select one of these...
Figure B-1: The Title Page of the Physical Geology Tutor
choices by clicking the mouse on the desired chapter name.

B.1.2 On-Line Instructions

The Using the Physical Geology Tutor option is an on-line instruction guide that familiarizes the user with the various commands for navigating the site. It also explains the functions available on the PGT menubars found on each Tutor page, as well as the various buttons used in the Quiz and Workbook sections. The on-line document contains much of the same information as this User's Guide.

B.1.3 PGT Chapters

There are eight chapters in the Physical Geology Tutor; each treats various geological features in the form of Textbook, Quiz and Workbook sections. Chapter choices include: Eolian, Fluvial, Glacial, Karst, Lacustrine, Shoreline, Structural Geology, and Vulcanism. Clicking on any of these chapters loads a page that gives the user the options within the chapter.

B.1.4 PGT Navigation

On each page there is at least one menubar that is used to navigate within the PGT (Figure B-2). The Next and Prev buttons move the user onto the next or the previous page respectively along the path prescribed by the PGT. The Chapter Menu button takes the user to the beginning of the chapter where the contents of the chapter are displayed. The Main Menu button takes the user back to the beginning of the Tutor where the available chapters are displayed. The Quiz button takes the user to the beginning of the Quiz for that chapter. The Workbook button takes the user to the beginning of the Workbook section for that chapter. The Index button brings up the Index for the chapter.
B.2 Textbook Section

The Textbook section of each PGT chapter contains definitions, descriptions, and illustrations of various geologic terms and concepts. When a chapter is selected from the main page, a Table of Contents page is loaded for that chapter. An example Table of Contents page is shown in Figure B-3. The menubar at the top and bottom of the pages may be used to progress through the Textbook. Additionally, each page within the Textbook is hyperlinked from the Table of Contents page, so simply clicking the mouse on the page name will cause the page to be loaded. Finally, if a student knows the exact name of the page she wishes to access, she may type the URL for that page in the browser window.

Textbook pages contain text definitions and descriptions of geologic features. An example Textbook page is shown in Figure B-4. The geologic term or concept described on the page is listed at the top of the page, and is also boldfaced the first time it appears in the text. Additionally, synonyms and other important terms are boldfaced. Terms which are defined on other pages are hyperlinked to those pages and appear underlined in the text. The student can click on the underlined terms to load the page where the term is defined.
3.3.1.4 Horn

A horn is a single remnant or high point of rock which forms when three or more cirques converge from different directions.

The Matterhorn.  
Original from the Ward Collection
He should click on the browser’s **Back** button to return to the original page he was reading. Many Textbook pages also contain one or more illustrations. When more than one picture is available, smaller versions of the additional pictures are displayed below the large image on the page. The student can click on one of the smaller pictures to reload the page with the full-sized version of the picture displayed.

### B.3 Quiz Section

After the student has read the entries in the Textbook section for a chapter, she can proceed to the **Quiz** section for the chapter. An example Quiz page is shown in Figure B-5. The

![Eolian Quiz](image)

**Figure B-5: A Single-Answer Quiz Page**

Quizzes are a series of multiple choice questions that have one or more correct answers. The Quizzes are implemented using CGI forms, so the student will click on a small checkbox to select an answer to submit. Only one checkbox can be selected at a time; if the student wishes to change an answer before submitting it, she simply needs to click the mouse on a different answer, and the first checkbox will be de-selected. After she has selected an answer, she should press the **Am I Right?** button to check her answer. If the answer is correct, the page will reload with a line stating that the answer is right; if the answer is
incorrect, the page will reload with a message encouraging the student to try again. The
student can then click on a new checkbox and resubmit her new answer with the Am I Right?
button. A number of Quiz questions have more than one correct answer\(^1\); since only one
checkbox can be checked at a time, the student will need to submit separately each answer
that she thinks is correct.

The Quizzes are not scored, and the answers are not collected anywhere on the webserver,
so it is to a student’s advantage to retry incorrectly answered questions until the correct
answer is submitted. The Quizzes range in length from 10 to over 220 questions, and it
is not necessary to finish all the questions in one session. A listing of question numbers is
given at the start of the Quiz, and if a student needs to stop in the middle of the Quiz, she
should remember what question number she completed so that she can resume the Quiz
from that point at a later time.

B.4 Workbook Section

The **Workbook Section** for each chapter\(^2\) consists of three modules: the **Name and
Outline** module, the **Name Only** module, and the **Outline Only** module.

The generic **Workbook** page displays a picture of a geological feature, various text
boxes and buttons that the student uses to submit his answers, and an underlined link
entitled **Look up feature** (Figure B-6). Students can click on the Look up feature link from
any Workbook page to go back to the Textbook+ page which describes the feature displayed
on that Workbook page. Depending on the type of exercise, the student may need to click
the points of a polygon over the picture to draw an outline, type in a text string in a text
entry box, click the mouse on one or more of the buttons, or any combination of the above
actions.

---

\(^1\)Questions with more than one correct answer are followed by a sentence stating that there is more one
correct answer.
\(^2\)The Lacustrine chapter does not have a Workbook section.
Figure B-6: Name and Outline Workbook Page (Horizontal Orientation)
B.4.1 Name and Outline Module

The **Name and Outline** module requires the most interaction from the student. In this part of the Workbook, the student must both draw a polygon around the feature shown in the picture and type the name of the feature. It does not matter which order the student does these two activities; the checking for the two activities is done independently.

For the Naming activity of the **Name and Outline** module, the student must name the feature shown in the picture. The student should click the mouse on the text field, which usually has a white background and is located next to the Submit Text button for horizontally-oriented pictures, and directly above the Submit Text button for vertically-oriented pictures, and then type in the answer. Once the student’s answer has been typed, he should click on the Submit Text button; pressing the Enter or Return key will not submit the answer. The Java applet will compare the student’s answer to the correct string, and display the word **Correct!** if the answer is correct or the word **Wrong!** if the answer is incorrect. Once a student has submitted an answer, he may get the correct answer by clicking on the Show Answer button; the answer will be displayed either next to or underneath the word **Answer.** If a student wishes to clear all of the text from the windows (both student-typed and computer-provided text), he should click on the Clear Text button.

In the Outlining activity the user has to outline the geological feature represented in the picture. The student must draw a polygon by clicking the mouse wherever he wishes to draw a vertex of the polygon. A line is drawn as the mouse is moved; there is no need to hold the mouse button down while moving the mouse. Clicking the mouse at a point creates a vertex of the polygon, and the line that will connect this vertex with the next one follows the mouse pointer around as it is moved. Once the polygon is drawn, the student should close the polygon by clicking the mouse on or near the original vertex. Once this has been done, the answer can be checked using the Check Outline button, which tells the user whether the answer he has submitted is right or wrong. The correct outline can be displayed by pressing the Show Outline button, and then moving the mouse pointer onto the picture. If the students wishes to draw a new outline, he can press the Clear Outline button and then move the mouse pointer over the picture. The outlines will disappear, and he can draw a new polygon and submit it.
B.4.2 Name Only Module

In the Name Only module, a picture is displayed with an outline around the feature that the student is to identify. The student must type in the name of the feature, similar to the Naming part of the Name and Outline module. Only the text areas and buttons used for the naming function will be present on the screen.

B.4.3 Outline Only Module

In the Outline Only module, a name is provided for the geological feature shown in the picture. The student must draw an outline around the feature, similar to the Outlining part of the Name and Outline module. Only the buttons and text boxes pertaining to the Outlining feature will be displayed on the screen.

B.5 Indices

Each chapter in the Physical Geology Tutor has an Index of the terms used within the Textbook sections (Figure B-7). Indexed words include terms which have Textbook pages dedicated to them, as well as other terms which are used as auxiliary terms in the Textbook pages. The terms are presented alphabetically in a list format, and the user can click on a selected term to load that page where the term is used. To return to the Index, the student should use the browser's Back button.

There is also an Index for the entire Physical Geology Tutor which can be accessed from the Main Page. This page includes all of the terms from the Chapter Indices listed alphabetically. Terms described in more than one chapter are listed twice, with the chapter name shown after the term.

B.6 Miscellaneous

A Photograph and Illustration References page recognizes the people whose photographs, illustrations, and textbooks have been used in the PGT.

A Questions, Comments, Ideas can be used by users of the Tutor to make comments about the Tutor, report bugs, and suggest additions or changes for the Tutor. The PGT
Figure B-7: An Index Page from the Physical Geology Tutor
Survey is a comment form that asks students for feedback about the Textbook, Quiz, and Workbook sections of the Tutor. It is a more formal way to return comments than sending mail from the Questions, Comments, Ideas page.

B.7 Error Problems and Solutions

Network Connection Refused error: The Physical Geology Tutor is currently supported on a private Athena workstation, and as such, when the machine is restarted, the webserver which hosts the PGT needs to be restarted by hand. Occasionally, the webserver is not restarted immediately after a reboot, so the PGT will not be accessible. If either of the messages shown in Figures B-8 or B-9 appears in the browser window, send mail to pgt-admin@mit.edu stating that the webserver appears to be down. The PGT Administrator will restart the server as soon as the message is received.

Forbidden error: Due to copyright restrictions on many of the photographs used in the PGT, access to the Tutor is limited to machines located within the mit.edu domain. If access is attempted from an outside site, the error message in Figure B-10 will be displayed. Plans are underway to obtain permission to grant outside access to the Physical Geology Tutor, but for now, the only way to view the Tutor is from a computer within the mit.edu domain.
Figure B-8: Network Error Message

A network error occurred:
unable to connect to server (TCP Error: Broken pipe)
The server may be down or unreachable.
Try connecting again later.

Figure B-9: Network Connection Refused Message

Netscape’s network connection was refused by the server:
cfd4.mit.edu
The server may not be accepting connections or
may be busy.
Try connecting again later.

Figure B-10: Forbidden/No Permission Message

Forbidden
You don’t have permission to access /PGTprivate/ on this server.
Appendix C

Example Textbook and Quiz Source Code

C.1 Example Textbook File: horn.html

```html
<html>
<head>
<title>Physical Geology Tutor: Glacial Features – Horn</title>
</head>

<body background="../../common/bkg.gif">

<a href="col.html">Prev</a> | <a href="hang_val.html">Next</a> | <a href="./">Chapter Menu</a> | <a href="../">Main Menu</a> | <a href="quiz/">Quiz</a> | <a href="workbook/">Workbook</a> | <a href="glaindex.html">Index</a><p>

<h1><a name="horn">3.3.1.4 Horn</a></h1>

A <strong>horn</strong> is a single remnant or high point of rock which forms when three or more <a href="cirque.html">cirques</a> converge from different directions.<p>

<img alt="glacier" src="pictures/glacier2.gif"><br>

Zumberge, J.H. and C.A. Nelson<br>
Elements of Physical Geology<br>
John Wiley &amp; Sons, Inc., New York, 1976<p>

<img alt="(glacier)" src="pictures/glacier2tiny.gif"><br>
<img alt="(horn)" src="pictures/glac_mnttiny.gif"><br>
<img alt="(Matterhorn)" src="pictures/LWID_2.12tiny.jpg"><br>
<img alt="(Red Needle Range)"

</body>
</html>
```
C.2 Example Single-Answer Quiz File: eolquiz46.cgi

#!/usr/athena/bin/perl

require "/var/home/etc/httpd/cgi-bin/cgi-lib.pl";
require "/var/home/etc/httpd/htdocs/PGTprivate/eolian/quiz/quiz.pl";

##
## Main
##

MAIN:
{
    local ($url, $url2);
    $url="eolquiz45.cgi";
    $url2="eolquiz47.cgi";

    if (&ReadParse("$input")) {
        &ProcessForm;
    } else {
        &SubmitForm;
    }
}

##
## Question -- trying with variables that are not initially there
## seems to work okay....

sub Question
{
    print <<EOT;
    <form method="post" action="eolquiz46.cgi">
    <h2>Question #46</h2>
    </form>
    <h2>Question #46</h2>
}
Which describes the downward slope of a transverse dune?

EOT

if ($input{'question'} eq "right1") {
    print <<EOT;
    <input type="radio" name="question" value="right1" checked>steep<br>
    EOT
}
else {
    print <<EOT;
    <input type="radio" name="question" value="right1">steep<br>
    EOT
}

if ($input{'question'} eq "wrong2") {
    print <<EOT;
    <input type="radio" name="question" value="wrong2" checked>gentle<br>
    EOT
}
else {
    print <<EOT;
    <input type="radio" name="question" value="wrong2">gentle <br>
    EOT
}

if ($input{'question'} eq "wrong3") {
    print <<EOT;
    <input type="radio" name="question" value="wrong3" checked>flat<br>
    EOT
}
else {
    print <<EOT;
    <input type="radio" name="question" value="wrong3">flat<br>
    EOT
}

if ($input{'question'} eq "wrong4") {
    print <<EOT;
    <input type="radio" name="question" value="wrong4" checked>moderate<br>
    EOT
}
else {
    print <<EOT;
    <input type="radio" name="question" value="wrong4">moderate<br>
    EOT
}

print "</font>";
}

## SubmitForm
##

sub SubmitForm
### C.3 Example Multiple-Answer Quiz File: eolquiz98.cgi

```
#!/usr/athena/bin/perl

require "/var/home/etc/httpd/cgi-bin/cgi-lib.pl";
require "/var/home/etc/httpd/htdocs/PGTprivate/eolian/quiz/quiz.pl";

#
## Main
#

MAIN: {
  local ($url, $url2);
  $url = "eolquiz97.cgi";
  $url2 = "eolquiz99.cgi";
  if (&ReadParse("input")) {
    &ProcessForm;
  } else {
    &SubmitForm;
  }
}
```
## Question — trying with variables that are not initially there
## seems to work okay....

sub Question
{
    print "<EOT;
<form method="post" action="eolquiz98.cgi">
<h2>Question #98</h2>
<font size=+1>
Which describes the channels/holes in loess deposits?<br>
(There are multiple correct answers)<p>
EOT

    if ($input{'question'} eq "wrong1") {
        print "<EOT;
        <input type="radio" name="question" value="wrong1" checked>inclined<br>
        EOT
    } else {
        print "<EOT;
        <input type="radio" name="question" value="wrong1">inclined<br>
        EOT
    }

    if ($input{'question'} eq "right2") {
        print "<EOT;
        <input type="radio" name="question" value="right2" checked>moderate depth<br>
        EOT
    } else {
        print "<EOT;
        <input type="radio" name="question" value="right2">moderate depth<br>
        EOT
    }

    if ($input{'question'} eq "right3") {
        print "<EOT;
        <input type="radio" name="question" value="right3" checked>vertical<br>
        EOT
    } else {
        print "<EOT;
        <input type="radio" name="question" value="right3">vertical<br>
        EOT
    }

    if ($input{'question'} eq "wrong4") {
        print "<EOT;
        <input type="radio" name="question" value="wrong4" checked>very deep<br>
        EOT
    } else {
        print "<EOT;
        <input type="radio" name="question" value="wrong4">very deep<br>

    print "<EOT;
</form>
";
}
if ($input{'question'} eq "wrong5") {
    print <<EOT;
<input type="radio" name="question" value="wrong5" checked>shallow<p>
EOT
}
else {
    print <<EOT;
<input type="radio" name="question" value="wrong5">shallow<p>
EOT
};

print "</font>";
}

## SubmitForm
##
sub SubmitForm
{
    &Header;
    &Question;
    &Submission;
    &Url;
    &Footer;
}

## ProcessForm
##
sub ProcessForm
{
    &Header;
    if (($input{'question'} eq "right2") || ($input{'question'} eq "right3")) {
        &Question;
        &SubForm;
        &Yep;
    } else {
        &Question;
        &SubForm;
        &Nope;
    }
    &Url;
    &Footer;
}

Appendix D

Template Textbook and Quiz Source Code

D.1 Textbook Template

<html>
<head>
<title>Physical Geology Tutor: PUT TITLE HERE</title>
</head>

<body background="../common/bkg.gif">

<h1><a name="FEATURENAME">PUT TITLE HERE (AGAIN)</a></h1>

<a href="PREVPAGE.HTML">&lt;- Prev</a> | 10 | <a href="NEXTPAGE.HTML">Next -&gt;</a> | 1
<a href="./">Chapter Menu</a> | 30 | <a href="../">Main Menu</a> | 20 | <a href="quiz/">Quiz</a> | 30 | <a href="workbook/">Workbook</a> | 20 | <a href="CHAINDEX.HTML">Index</a> | 1

<p>
<font size=+1>

TYPE IN THE DESCRIPTION FOR THE CHAPTER HERE. INCLUDE THE TEXT, AS WELL AS ANY PICTURES OR ILLUSTRATIONS THAT ARE NECESSARY. MAKE SURE THAT THE DEFINED WORD IS IN BOLDFACE. LOOK AT EXISTING FILES FOR THE CORRECT FORMAT FOR ADDING PICTURES AND HYPERLINKS.

</font></p>

</body>
</html>
D.2 Single-Answer Quiz Template

#!/usr/athena/bin/perl

require "/var/home/etc/httpd/cgi-bin/cgi-lib.pl";
require "/var/home/etc/httpd/htdocs/PGTprivate/CHAPTERNAME/quiz/quiz.pl";
## CHAPTERNAME in the above line should be changed to the appropriate
## lowercase chapter name:
## require "/var/home/etc/httpd/htdocs/PGTprivate/glacial/quiz/quiz.pl";
##
### Main
###
MAIN:
{
## Replace CHA with the lowercase first three letters of a chapter title:
## eol, flu, gla, kar, lac, sho, str, vul
## Replace (x-1) and (x+1) with the previous and next quiz question numbers.
##
## For example, Eolian quiz question #7, the $url and $url2 lines should read:
## $url="eolquiz106.cgi";
## $url2="eolquiz108.cgi";

local ($url, $url2);
$url="CHAquiz(x-1).cgi";
$url2="CHAquiz(x+1).cgi";

if (&ReadParse(*input)) {
  &ProcessForm;
} else {
  &SubmitForm;
}
}
##
## Question -- trying with variables that are not initially there
## seems to work okay....

sub Question
{
## Again, replace CHA and (x) as above.

Which describes the downward slope of a transverse dune?

```html
<form method="post" action="CHAquiz\(x)\).cgi">
<h2>Question #\(x\)</h2>
<font size=+1>
Which describes the downward slope of a transverse dune?<p>
</font>
</form>

### For this sequence of if/else statements, replace the string that comes before the <br> tag with the appropriate answer. The answer that is incorrect should say "wrong\(N\)", where \(N\) is the answer number. Answers that are correct should say "right\(N\)". There should only be one correct answer. If there is more than one correct answer, use the "quizmultiple.cgi" template instead. If there are more than the shown number of answers, add as many more if/else pairs as needed. Likewise, if there are fewer answers to the question than there are shown, delete as many if/else pairs as needed.

```perl
if ($input{'question'} eq "right1") {
    print <<EOT;
    <input type="radio" name="question" value="right1" checked>ANSWER 1<br>
EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="right1">ANSWER 1<br>
EOT
}
```

```perl
if ($input{'question'} eq "wrong2") {
    print <<EOT;
    <input type="radio" name="question" value="wrong2" checked>ANSWER 2<br>
EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="wrong2">ANSWER 2<br>
EOT
}
```

```perl
if ($input{'question'} eq "wrong3") {
    print <<EOT;
    <input type="radio" name="question" value="wrong3" checked>ANSWER 3<br>
EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="wrong3">ANSWER 3<br>
EOT
}
```

```perl
if ($input{'question'} eq "wrong4") {
    print <<EOT;
    <input type="radio" name="question" value="wrong4" checked>ANSWER 4<br>
EOT
```
D.3 Multiple-Answer Quiz Template

#!/usr/athena/bin/perl
require "/var/home/etc/httpd/cgi-bin/cgi-lib.pl";
require "/var/home/etc/httpd/htdocs/PGTprivate/CHAPTERNAME/quiz/quiz.pl";

## CHAPTERNAME in the above line should be changed to the appropriate
## lowercase chapter name:
## require "/var/home/etc/httpd/htdocs/PGTprivate/glacial/quiz/quiz.pl";

# # Main
#
MAIN:
{

## Replace CHA with the lowercase first three letters of a chapter title:
## eol, flu, gla, kar, lac, sho, str, vul
## Replace (x-1) and (x+1) with the previous and next quiz question numbers.
##
## For example, Eolian quiz question #7, the $url and $url2 lines should read:
##
## $url="eolquiz106.cgi";
## $url2="eolquiz108.cgi";

local ($url, $url2);
$url="CHAquiz(x-1).cgi";
$url2="CHAquiz(x+1).cgi";

if (&ReadParse(*input)) {
  &ProcessForm;
} else {
  &SubmitForm;
}

## Question -- trying with variables that are not initially there
## seems to work okay....

sub Question
{
  ## Again, replace CHA and (x) as above.
  print <<EOT;
  <form method="post" action="CHAquiz(x).cgi">
  <h2>Question #(x)</h2>
  <font size=+1>
  (This line of text needs to be replaced with the question text.)?<br>
  (There are multiple correct answers)<p>
  EOT

  ## For this sequence of if/else statements, replace the string that
  ## comes before the <br> tag with the appropriate answer. Answers
  ## that are incorrect should say "wrongN", where N is the answer
  ## number. Answers that are correct should say "rightN". If there is
  ## only one correct answer, use the "quizsingle.cgi" template instead.
  ## If there are more than the shown number of answers, add as many
  ## more if/else pairs as needed. Likewise, if there are fewer answers
### to the question than there are shown, delete as many if/else pairs
### as needed.

```perl
if ($input{'question'} eq "wrong1") {
    print <<EOT;
    <input type="radio" name="question" value="wrong1" checked>ANSWER 1<br>
    EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="wrong1">ANSWER 1<br>
    EOT
}

if ($input{'question'} eq "right2") {
    print <<EOT;
    <input type="radio" name="question" value="right2" checked>ANSWER 2<br>
    EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="right2">ANSWER 2<br>
    EOT
}

if ($input{'question'} eq "right3") {
    print <<EOT;
    <input type="radio" name="question" value="right3" checked>ANSWER 3<br>
    EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="right3">ANSWER 3<br>
    EOT
}

if ($input{'question'} eq "wrong4") {
    print <<EOT;
    <input type="radio" name="question" value="wrong4" checked>ANSWER 4<br>
    EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="wrong4">ANSWER 4<br>
    EOT
}

if ($input{'question'} eq "wrong5") {
    print <<EOT;
    <input type="radio" name="question" value="wrong5" checked>ANSWER 5<br>
    EOT
} else {
    print <<EOT;
    <input type="radio" name="question" value="wrong5">ANSWER 5<br>
    EOT
}
```
print "</font>");

##
## SubmitForm
##

sub SubmitForm
{
    &Header;
    &Question;
    &Submission;
    &Url;
    &Footer;
}

##
## ProcessForm
##

sub ProcessForm
{
    &Header;

    ## The correct question numbers should be listed here. If there are
    ## more than two correct answer, add another "|| ($input{ 'question'}
    ## eq "rightN")" set.
    ##
    ## For example:
    ## if (($input{ 'question'} eq "right2") || ($input{ 'question'} eq "right3")
    ##   || ($input{ 'question'} eq "right4")) {
    ##
    ##      if (($input{ 'question'} eq "right2") || ($input{ 'question'} eq
    ##        "right3")
    ##        {
    ##          &Question;
    ##          &SubForm;
    ##          &Yep;
    ##        }
    ##      else {
    ##        &Question;
    ##        &SubForm;
    ##        &Nope;
    ##      }
    ##}
    ##
    &Url;
    &Footer;
}
Appendix E

Workbook Applet Source Code

E.1 Both.java

// Both.java

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.util.*;
import java.lang.*;

public class Both extends Applet {

    // Data used by main class
    int correct_outline; // The image used; parameter gotten from the HTML file
    Polygon innerpoly = new Polygon(); // Inner part of bandwidth polygon
    Polygon outerpoly = new Polygon(); // Outer part of bandwidth polygon

    private boolean answered = false; // Has an answer been submitted yet?
    private BothCan bothcanvas; // The canvas where the cool stuff happens
    private int imgheight; // image height
    private int imgwidth; // image width
    private String answerstring; // The correct answer
    private String dummyx; // dummy variable for tokenizing x-coordinates
    private String dummyy; // dummy variable for tokenizing y-coordinates
    private String name; // The name of the image (ends in .jpg)
    private int xinstring; // x-coordinates of inner polygon
    private int xoutstring; // x-coordinates of outer polygon
    private int yinstring; // y-coordinates of inner polygon
    private int youtstring; // y-coordinates of outer polygon
    private TextField answer; // The field for the correct answer
    private TextField rightorwrong; // Where to show "correct" for outlines
    private TextField sol; // The field for the correct answer
    private TextField youranswer; // The field for the student's answer

    private void add (Component c, GridBagLayout gbl, GridBagConstraints gbc, int x, int y, int w, int h) {
        // ******************** add method (overloaded) ********************
        // ******************** Main initialisation method ********************

        // ********************
        // ********************

    }

gbc.gridx = x;
gbc.gridy = y;
gbc.gridwidth = w;
gbc.gridheight = h;
gbl.setConstraints(c, gbc);
add(c);
}

/********************* init method *******************/
/**************** Main initialisation method ***********/

public void init() {
    /* Get parameters from the html page */
    answerstring = getParameter(" featurename");
    name = getParameter(" img");
xinstring = getParameter(" innerx").trim();
yinstring = getParameter(" inny").trim();
xoutstring = getParameter(" outerx").trim();
youtstring = getParameter(" outery").trim();
String dummyw = getParameter(" iwidth").trim();
String dummyh = getParameter(" iheight").trim();
imgwidth = Integer.parseInt(dummyw);
imgheight = Integer.parseInt(dummyh);

    /* Tokenize the data for the bandwidth polygon coordinates */
    StringTokenizer tix = new StringTokenizer(xinstring, ", ");
    StringTokenizer tiy = new StringTokenizer(yinstring, ", ");
    StringTokenizer tox = new StringTokenizer(xoutstring, ", ");
    StringTokenizer toy = new StringTokenizer(youtstring, ", ");

    int ptsin = tix.countTokens();
    int[] innerx = new int[ptsin];
    int[] inny = new int[ptsin];
    int[] outerx = new int[ptsout];
    int[] outery = new int[ptsout];

    for(int i=0; i<ptsin; i++) {
        dummyx = tix.nextToken();
dummyy = tiy.nextToken();
        System.out.println(dummyx + ", " + dummyy + "\n");
        innerx[i] = Integer.parseInt(dummyx);
inny[i] = Integer.parseInt(dummyy);
    }

    for(int j=0; j<ptsout; j++) {
        dummyx = tox.nextToken();
dummyy = toy.nextToken();
        System.out.println(dummyx + ", " + dummyy + "\n");
        outerx[j] = Integer.parseInt(dummyx);
outery[j] = Integer.parseInt(dummyy);
    }

    Polygon innerpoly = new Polygon(innerx, inny, ptsin); /* in band poly */
    Polygon outerpoly = new Polygon(outerx, outery, ptsout); /* out band poly */

    /* Setup grid */
}
GridBagLayout gbl = new GridBagLayout();
setLayout(gbl);

String pictureUrl = "images/" + name;
picture = getImage(getDocumentBase(), pictureUrl);
bothcanvas = new BothCan(picture, imgwidth, imgheight, innerx, inners, outerx, outery, innerpoly, outerpoly);
bothcanvas.resize(imgwidth, imgheight);
youranswer = new TextField("", 40);
Button submitButton = new Button("Submit Text");
Button clearButton = new Button("Clear Text");
sol = new TextField("", 15);
sol.setEditable(false);
Button showButton = new Button("Show Answer");
Label label = new Label("Answer:");
answer = new TextField("", 50);
answer.setEditable(false);
Panel blankPanel = new Panel();
Button showOut = new Button("Show Outline");
Button rightOut = new Button("Check Outline");
Button clearOut = new Button("Clear Outline");
rightorwrong = new TextField("", 50);
rightorwrong.setEditable(false);

/* Horizontal layout for landscape-oriented pictures */
if (imgwidth > imgheight) {
    Panel horiz1 = new Panel();
    horiz1.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz1.add(youranswer);
    horiz1.add(submitButton);

    Panel horiz2 = new Panel();
    horiz2.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz2.add(label);
    horiz2.add(answer);

    Panel horiz3 = new Panel();
    horiz3.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz3.add(sol);

    Panel horiz4 = new Panel();
    horiz4.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz4.add(showButton);
    horiz4.add(clearButton);

    Panel horiz5 = new Panel();
    horiz5.setLayout(new FlowLayout(FlowLayout.CENTER));
    horiz5.add(rightOut);
    horiz5.add(showOut);
    horiz5.add(clearOut);

    Panel horiz6 = new Panel();
    horiz6.add(rightorwrong);
}
Panel horiz7 = new Panel();
horiz7.setLayout(new GridLayout(2,1));
horiz7.add(horiz5);
horiz7.add(horiz6);

Panel horizPanel = new Panel();
horizPanel.resize(imgwidth, 120);
horizPanel.setLayout(new BorderLayout());
horizPanel.add("North", horiz1);
horizPanel.add("South", horiz2);
horizPanel.add("West", horiz3);
horizPanel.add("East", horiz4);

Panel outPanel = new Panel();
outPanel.resize(imgwidth,(imgheight+200));
outPanel.setLayout(new BorderLayout());
outPanel.add("North", horizPanel);
outPanel.add("South", horiz7);

GridBagConstraints horiz = new GridBagConstraints();
horiz.fill = GridBagConstraints.BOTH;
horiz.anchor = GridBagConstraints.NORTHWEST;
horiz.weightx = 0;
horiz.weighty = 0;
add(bothcanvas, gbl, horiz, 0, 0, 5, 6);

horiz.fill = GridBagConstraints.NONE;
horiz.anchor = GridBagConstraints.SOUTHWEST;
horiz.weighty = 100;
horiz.weightx = 100;
add(outPanel, gbl, horiz, 0, 6, 5, 3);
}

/* Vertical layout for portrait-oriented pictures */
else {

Panel vert1 = new Panel();
vert1.setLayout(new FlowLayout(FlowLayout.CENTER));
vert1.add(submitButton);
vert1.add(clearButton);

Panel vert2 = new Panel();
vert2.setLayout(new GridLayout(5,1));
vert2.add(answer);
vert2.add(label);
vert2.add(rightOut);
vert2.add(showOut);
vert2.add(clearOut);
vert2.add(rightorwrong);
vert2.resize(220, imgheight);

Panel vert3 = new Panel();
vert3.setLayout(new FlowLayout(FlowLayout.CENTER));
vert3.add(showOut);
vert3.add(clearOut);

Panel vertPanel = new Panel();
vertPanel.setLayout(new GridLayout(10,1));
vertPanel.add(youranswer);
vertPanel.add(vert1);
vertPanel.add(sol);
vertPanel.add(showButton);
vertPanel.add(label);
vertPanel.add(ans);
vertPanel.add(rightOut);
vertPanel.add(vert3);
vertPanel.add(rightorwrong);
vertPanel.resize(230, imgheight);

GridBagConstraints vert = new GridBagConstraints();
vert.fill = GridBagConstraints.BOTH;
vert.anchor = GridBagConstraints.NORTHWEST;
vert.weightx = 0;
vert.weighty = 0;
add(bothcanvas, gbl, vert, 0, 0, 2, 2);
vert.fill = GridBagConstraints.NONE;
vert.anchor = GridBagConstraints.NORTHEAST;
vert.weighty = 100;
vert.weightx = 100;
add(vertPanel, gbl, vert, 2, 0, 1, 1);
vert.anchor = GridBagConstraints.SOUTHEAST;
add(blankPanel, gbl, vert, 2, 1, 1, 1);
resize((imgwidth+230),imgheight);

}
else {
    rightorwrong.setText("Finish drawing the polygon.");
    //
    System.out.println("Finish drawing & then check.");
}
else if (arg.equals("Clear Outline")) {
    bothcanvas.clearvariables();
    rightorwrong.setText("");
} else if (arg.equals("Submit Text")) {
    answered = true;
    //
    System.out.println("subtext answered = " + answered + "\n");
    String youranswerstring = youranswer.getText();
    if (youranswerstring.equals(answerstring)) {
        sol.setText("Correct! ");
    } else {
        sol.setText("Wrong! ");
    }
} else if (arg.equals("Show Answer")) {
    if (answered == true) {
        answer.setText (answerstring);
    } else {
        answer.setText("Submit an answer first.");
    }
    //
    System.out.println("answered: " + answered + "\n");
}
else if (arg.equals("Clear Text")) {
    youranswer.setText("'";
    answer.setText(" ");
    sol.setText(" ");
    answered = false;
} else return super.action(evt, arg);
return true;
}
/**
 ************************************************************
 BothCan.class **************************************************
 **** This class is where the image is displayed on a canvas ******
*******************************************************************/

class BothCan extends Canvas {
    final int MAXLINES = 30;  /* Max number of lines */
    final int TOLERANCE = 8;  /* How close before snapping to */
    boolean done = false;     /* Am I done drawing the polygon? */
    boolean rightoutline = false; /* Is the student polygon correct? */
    boolean showoutline = false; /* Do we show the correct outline? */
    final double FRACTION_ALLOWED = .4; /* Allowable fraction of vertices */
* (that can be outside ref.
* polygons) */

Image imagine; /* Local name for image object */
int beginx; /* The x coord of the start point */
int beginy; /* The y coord of the start point */
int currline = 0; /* number of lines */
int currx;
int curry;
int imgheight;
int imgwidth;
int[] inax;
int[] inay;
int[] outax;
int[] outay;

int[] student_x = new int[MAXLINES + 1]; /* x-coords of drawn polygon */
int[] student_y = new int[MAXLINES + 1]; /* y-coords of drawn polygon */
int[] xinner = new int[MAXLINES]; /* Reference Polygon outer & inner */
int[] yinner = new int[MAXLINES]; /* vertices. Though I don't have */
int[] xouter = new int[MAXLINES]; /* have to configure the polygons */
int[] youter = new int[MAXLINES]; /* in the main class now */

Point vertices[] = new Point[MAXLINES + 1]; /* starting points */
Point dummy[] = new Point[MAXLINES + 1]; /* starting points */

Point anchor = null; /* start of current line */
Point oldanchor = null; /* start of previous line */
Point currentpoint = null; /* current end of line */
Point beginning = null; /* an alias for int = vertices[0] */
Polygon ipoly; /* The inner and outer bandwidth */
Polygon opoly; /* polygon vertices */

/**************************** BothCan Constructor Methods ****************************/
/**************************** Set up the damn Constructor Method with ****************************
/**************************** the type of variables it can take! ****************************/

public BothCan(Image i) {
    imagine = i;
}

public BothCan(Image i, int inax, int inay, int outax, int outay) {
    imagine = i;
}

public BothCan(Image i, Polygon p, Polygon q) {
    imagine = i;
    ipoly = p;
    opoly = q;
}

public BothCan(Image i, int w, int h, int[] ix, int[] iy, int[] ox,
    int[] oy, Polygon p, Polygon q) {
    imagine = i;
    imgheight=w;
    imgwidth=w;
inax = ix;
inay = iy;
outax = ox;
outay = oy;
ipoly = p;
oapy = q;
resize(imgwidth, imgheight);
}

/* Read the down—click of the
* mouse & deal with that data */

/**/ mouseDown method (overridden) *********
/** overrides standard mouseDown method **********/
public boolean mouseDown(Event evt, int x, int y) {
    anchor = new Point(x, y);
    if (currline == 0) {
        beginning = anchor; /* This point is the starting point */
        beginx = x;
        beginy = y;
    }
    if (done) {
        System.out.println("You’re done drawing the polygon!");
    } else if (currline <= (MAXLINES - 1))
        addLine(x, y);
    // else if (currline == (MAXLINES-1))
    // addLine(vertices[0].x,vertices[0].y);
    else System.out.println("Too many lines");
    // System.out.println("currline=" + currline + " beginning= " + beginning);
    return true;
}

/* Deal with the movement of the mouse */

/**/ mouseMove method (overridden) *********
/** overrides standard mouseMove method **********/
public boolean mouseMove(Event evt, int x, int y) {
    if (anchor == null)
        currentpoint = anchor;
    else
        currentpoint = new Point(x, y);
    repaint();
    return true;
}

/**/ addLine method *********
/** Keep track of the number of lines that’ve been drawn **********/
public void addLine(int x, int y) {
    if (currline == (MAXLINES - 1)) {
        vertices[currline] = beginning;
        student_x[currline] = beginx;
        student_y[currline] = beginy;
        student_x[(currline+1)] = -1;
        student_y[(currline+1)] = -1;
        done = true;
    } else if (((currline > 0) && currline < (MAXLINES - 1)) &&
            ((currentpoint.x > (beginning.x - TOLERANCE)) &&
            (currentpoint.x < (beginning.x + TOLERANCE))) &&
            ((currentpoint.y > (beginning.y - TOLERANCE)) &&
            (currentpoint.y < (beginning.y + TOLERANCE))) &&
            ((currentpoint.x > (beginning.x - TOLERANCE)) &&
            (currentpoint.x < (beginning.x + TOLERANCE))) &&
            ((currentpoint.y > (beginning.y - TOLERANCE)) &&
            (currentpoint.y < (beginning.y + TOLERANCE))) &&
            (currline < MAXLINES - 1)) {
        vertices[currline+1] = currentpoint;
        if (beginning == null)
            beginning = currentpoint;
        student_x[currline+1] = currentpoint.x;
        student_y[currline+1] = currentpoint.y;
        student_x[(currline+2)] = -1;
        student_y[(currline+2)] = -1;
        done = true;
    }
(currentpoint.y > (beginning.y - TOLERANCE)) \&\&
(currentpoint.y < (beginning.y + TOLERANCE)))
{
vertices[currline] = beginning;
student_x[currline] = beginx;
student_y[currline] = beginy;
student_x[(currline+1)] = -1;
student_y[(currline+1)] = -1;

// anchor = (beginning);
done = true;
}

else {
vertices[currline] = anchor;
student_x[currline] = x;
student_y[currline] = y;
}

// System.out.println("vertex[" + currline + "] = " + vertices[currline]);
currentpoint = vertices[currline];
currline++;
repaint();
}

/*********************** bandwidthInit method *********************/
/** Set up the arrays with the bandwidth polygon vertices */
public void bandwidthInit() {
System.arraycopy(inax, 0, xinner, 0, (inax.length));
System.arraycopy(inay, 0, yinner, 0, (inay.length));
System.arraycopy(outax, 0, xouter, 0, (outax.length));
System.arraycopy(outay, 0, youter, 0, (outay.length));
xinner[(inax.length)] = -1;
yinner[(inax.length)] = -1;
xouter[(outax.length)] = -1;
youter[(outay.length)] = -1;
}

/*********************** showCorrect method *********************/
/** Let the paint method know it's time to draw the correct outline */
public boolean showCorrect() {
if (done) {
showoutline=true;
return true;
}
else
return false;
}

/*********************** rightWrong method *********************/
/** Press a button to show the user if they're right or not */
public int rightWrong() {
/* Add the -1 to the arrays */
bandwidthInit();

if (done) {
rightoutline = checkPolygon();
if (rightoutline)
return 1;
else
    return 2;
else return 3;
}

/*********************** clearVariables method *********************/
/****************** Clears the variables so you can draw them all & try again **********/
public void clearvariables() {
    done = false;
    for (int j = 0; j < (MAXLINES + 1); j++) {
        vertices[j] = new Point(0,0);
        student_x[j] = 0;
        student_y[j] = 0;
    }
    showoutline = false;
    anchor = null;
    oldanchor = null;
    currentpoint = null;
    beginning = null;
    beginx = 0;
    beginy = 0;
    currline = 0;
}

/**************** Get rid of screen flicker */
/*********************** update method (overridden) *********************/
public void update(Graphics g) {
    paint(g);
}

/*********************** paint method (overridden) *********************/
/****************** Draw the image, the existing & current lines, & maybe the answer ****/
public void paint(Graphics g) {
    int imageWidth = imagine.getWidth(this);
    int imageHeight = imagine.getHeight(this);
    g.drawImage(imagine, 0,0, this);
    // Draw reference polygon if it's okay to do so *
    if (showoutline) {
        g.setColor(Color.red);
        g.drawPolygon(ipoly);
        g.drawPolygon(opoly);
    }
    // Draw existing lines *
    for (int i=1; i < currline; i++) {
        g.setColor(Color.yellow);
        g.drawLine(vertices[i-1].x, vertices[i-1].y, vertices[i].x, vertices[i].y);
    }
    // draw current line */
    g.setColor(Color.blue);
/ System.out.println("done = "+done+ "; currentpoint = "+currentpoint+"\n");
if (done);
else if (currline == MAXLINES);
else if (currentpoint != null) & (currline < MAXLINES)) {

/*
 * This bit of code snaps the end of the line to the first point if
 * the end vertex is close enough.  Grabbed from Pat's polyfuncs.c
 * code.
*/
if (currentpoint.x > (beginning.x - TOLERANCE) &&
    currentpoint.x < (beginning.x + TOLERANCE) &&
    currentpoint.y > (beginning.y - TOLERANCE) &&
    currentpoint.y < (beginning.y + TOLERANCE))
g.drawLine(anchor.x, anchor.y, beginning.x, beginning.y);
else
    g.drawLine(anchor.x, anchor.y, currentpoint.x, currentpoint.y);
}

/****************************findArea method ****************************/
/****** Finds the area of the polygons (template) *******/
public static int findArea(int xl, int yl, int x2, int y2, int x3, int y3) {
    int findarea = ((-x1*y2 + xl*y3 + x2*yl - x2*y3 - x3*yl + x3*y2));
    // if(findarea < 0)
    // findarea = -1 * findarea;
    return findarea;
}

/****************************checkPolygon method ****************************/
/***** Checks to see if polygon resembles answer (bandwidth) *******/
public boolean checkPolygon() {
    int stud_area; /* Area of student polygon */
    int num_wrong_vert = 0; /* Number of vertices outside of bandwidth */
    int j; /* Dummy counter */
    int foo;
    int bar;
    double divider;

    /* Calculate area of student polygon. */
    stud_area = polygonArea(student_x, student_y);
    System.out.println("stud area = " + stud_area + "\n");
    foo = polygonArea(xinner, yinner);
    bar = polygonArea(xouter, youter);
    System.out.println("ipolyarea = " + foo + " opolyarea = " + bar + "\n");

    /*
    * To avoid decimal errors, the areas are represented as twice the number
    * of pixels.
    */

    /*
    * Polygon vertex is wrong if student area does not have a value that
    * lies between inner and outer polygons.
    */
    if ((stud_area < polygonArea(xinner, yinner)) ||
        (stud_area > polygonArea(xouter, youter))) {

/* Polygon fails from area criteria. */
System.out.print("check_polygon: polygon is incorrect: wrong size\n");
return false;
}

/*
 * Since area meets requirement: Check whether student's polygon is
 * inside bandwidth polygon.
 */
j=1;  /* Counter variable. */

/* Checks all student vertices except the first because it's the last one. */
while (student_x[j] != -1) {
    if (ipoly.inside(student_x[j], student_y[j])) {
        ++num_wrong_vert;  /* Wrong: Vertex inside inner bandwidth. */
        // System.out.println("bad in; num_wrong_vert = " + num_wrong_vert + \n");
    }
    /*
     * Next, for each vertex, if it's outside the outer polygon, then
     * it's wrong.
     */
    else if (!opoly.inside(student_x[j], student_y[j])) {
        ++num_wrong_vert;  /* Wrong: Vertex outside outer bandwidth. */
        // System.out.println("bad out; num_wrong_vert = " + num_wrong_vert + \n");
    }
    ++j;  /* Increment index -> Check next student vertex. */
}

/* Calculate the fraction of vertices outside the bandwidth */
double divider3 = (double) num_wrong_vert / (j-1);
// System.out.println("divider3 = " + divider3 + "; j = " + j + "; num_wrong_vert = " +num_wrong_vert+ \n");
if (((double) num_wrong_vert / (j-1)) >= FRACTION_ALLOWED) {
/* Polygon fails from vertex criteria. */
    // System.out.print("check_polygon: polygon is incorrect: bad vertices\n");
    System.out.print("Percent wrong: " + ( (double) num_wrong_vert / (j-1)) + \n");
    return false;
}
else {
    /* Polygon is correct */
    System.out.print("Percent wrong: " + ((double) num_wrong_vert / (j-1)) + \n");
    // System.out.print("check_polygon: polygon is correct\n");
    return true;
}
}

******************************************************************************
polygonArea method ***************************************************************************/
***** Returns twice the number of pixels circumscribed by polygon *****
public static int polygonArea(int[] x, int[] y) {
    int poly_area = 0;  /* Initialization of Area */
    int j = 0;  /* Dummy Counter */
/ * If these are less than 3 vertices, the area is zero */
if (x[0] == (-1) || x[1] == (-1))
    return (0);
while (x[j+2] != (-1)) {
    poly_area += findArea(x[0], y[0], x[j+1], y[j+1], x[j+2], y[j+2]);
    ++j;
}
if (poly_area > 0)
    return (poly_area);
else
    return (-poly_area);
}

E.2 Name.java

// Name.java
import java.awt.*;
import java.applet.*;
import java.awt.image.*;
import java.util.*;
import java.lang.*;
public class Name extends Applet {
    public void paint(Graphics g) {
        // Draw the inner polygon
        g.setColor(Color.blue);
        g.fillPolygon(innerpoly);

        // Draw the outer polygon
        g.setColor(Color.red);
        g.drawPolygon(outerpoly);
    }

    // Method to find the area of a polygon
    private double findArea(double x[], double y[], int n) {
        double area = 0;
        for (int i = 0; i < n; i++) {
            int j = (i + 1) % n;
            area += (x[i] + x[j]) * (y[i] - y[j]);
        }
        return Math.abs(area / 2.0);
    }
}

private boolean answered = false; /* Has an answer been submitted yet? */
private int imgheight; /* image height */
private int imgwidth; /* image width */
private NameCanvas outcanvas; /* Canvas with image + outlines */
private String answerstring; /* The correct answer */
private String dummyx; /* dummy variable for tokenizing x-coordinates */
private String dummyy; /* dummy variable for tokenizing y-coordinates */
private String name; /* The name of the image (ends in .jpg) */
private String xinstring; /* x-coordinates of inner polygon */
private String xoutstring; /* x-coordinates of outer polygon */
private String yinstring; /* y-coordinates of inner polygon */
private String youtstring; /* y-coordinates of outer polygon */
private TextField answer; /* The field for "right" or "wrong" */
private TextField sol; /* The field for the correct answer */
private TextField youranswer; /* The field for the student's answer */

private void add(Component c, GridBagConstraints gbc, int x, int y, int w, int h) {
    gbc.gridx = x;
    gbc.gridy = y;
}
gbc.gridwidth = w;
gbc.gridheight = h;
gbl.setConstraints(c, gbc);
add(c);
}

/*********************** init method ***********************
/******* Main initialisation method *******
public void init() {

/* Get parameters from the html page */
answerstring = getParameter("featurename");
name = getParameter("img");
xinstring = getParameter("innerx").trim();
yinstring = getParameter("innyer").trim();
xoutstring = getParameter("outerx").trim();
youtstring = getParameter("outery").trim();
String dummyw = getParameter("iwidth").trim();
String dummyh = getParameter("iheight").trim();
imgwidth = Integer.parseInt(dummyw);
imgheight = Integer.parseInt(dummyh);

/* Tokenize the data for the bandwidth polygon coordinates */
StringTokenizer tix = new StringTokenizer(xinstring, ",");
StringTokenizer tiy = new StringTokenizer(yinstring, ",");
StringTokenizer tox = new StringTokenizer(xoutstring, ",");
StringTokenizer toy = new StringTokenizer(youtstring, ",");
int ptsin = tix.countTokens();
int ptsout = tox.countTokens();
int[] innerx = new int[ptsin];
int[] innyer = new int[ptsin];
int[] outerx = new int[ptsout];
int[] outery = new int[ptsout];
for(int i=0; i<ptsin; i++) {
    dummyx = tix.nextToken();
    dummyy = tiy.nextToken();
    // System.out.println(dummyx + "," + dummyy + 
    innerx[i] = Integer.parseInt(dummyx);
    innyer[i] = Integer.parseInt(dummyy);
}
for(int j=0; j<ptsout; j++) {
    dummyx = tox.nextToken();
    dummyy = toy.nextToken();
    // System.out.println(dummyx + "," + dummyy + 
    outerx[j] = Integer.parseInt(dummyx);
    outery[j] = Integer.parseInt(dummyy);
}
Polygon innerpoly = new Polygon(innerx, innyer, ptsin); /* in band poly */
Polygon outerpoly = new Polygon(outerx, outery, ptsout); /* out band poly */

/* Set up the layout of the applet */
GridBagLayout gbl = new GridBagLayout();
setLayout(gbl);

String pictureUrl = "images/" + name;

picture = getImage(getDocumentBase(), pictureUrl);
outcanvas = new NameCan(picture, imgwidth, imgheight, innerpoly, outerpoly);
youranswer = new TextField("", 40);
submitButton = new Button("Submit Text");
clearButton = new Button("Clear Text");
sol = new TextField("", 15);
sol.setEditable(false);
showButton = new Button("Show Answer");
label = new Label("Answer: ");
answer = new TextField("", 50);
answer.setEditable(false);
blankPanel = new Panel();

/* Horizontal layout for landscape-oriented pictures */
if (imgwidth > imgheight) {
    Panel horiz1 = new Panel();
    horiz1.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz1.add(youranswer);
    horiz1.add(submitButton);
    Panel horiz2 = new Panel();
    horiz2.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz2.add(label);
    horiz2.add(answer);
    Panel horiz3 = new Panel();
    horiz3.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz3.add(sol);
    Panel horiz4 = new Panel();
    horiz4.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz4.add(showButton);
    horiz4.add(clearButton);
    horizPanel = new Panel();
    horizPanel.setConstraints(new BorderLayout());
    horizPanel.add("North", horiz1);
    horizPanel.add("South", horiz2);
    horizPanel.add("West", horiz3);
    horizPanel.add("East", horiz4);
    GridBagConstraints horiz = new GridBagConstraints();
    horiz.fill = GridBagConstraints.BOTH;
    horiz.anchor = GridBagConstraints.NORTHWEST;
    horiz.weightx = 100;
    horiz.weighty = 100;
    add(outcanvas, gbl, horiz, 0, 0, 5, 6);
    horiz.fill = GridBagConstraints.NONE;
    horiz.anchor = GridBagConstraints.SOUTHWEST;
horiz.weighty = 0;
horiz.weightx = 0;
add(horizPanel, gbl, horiz, 0, 6, 5, 3);
resize(imgwidth, (imgheight + 120));
}

/* Vertical layout for portrait-oriented pictures */
else {
    Panel vert1 = new Panel();
    vert1.setLayout(new FlowLayout(FlowLayout.CENTER));
    vert1.add(submitButton);
    vert1.add(clearButton);
    Panel vertPanel = new Panel();
    vertPanel.resize((imgwidth+220), imgheight);
    vertPanel.setLayout(new GridLayout(6,1));
    vertPanel.add(youranswer);
    vertPanel.add(vert1);
    vertPanel.add(sol);
    vertPanel.add(showButton);
    vertPanel.add(label);
    vertPanel.add(answer);
    GridBagConstraints vert = new GridBagConstraints();
    vert.fill = GridBagConstraints.BOTH;
    vert.anchor = GridBagConstraints.NORTHWEST;
    vert.weightx = 100;
    vert.weighty = 100;
    add(outcanvas, gbl, vert, 0, 0, 2, 2);
    vert.fill = GridBagConstraints.NONE;
    vert.anchor = GridBagConstraints.NORTHEAST;
    vert.weighty = 0;
    vert.weightx = 0;
    add(vertPanel, gbl, vert, 2, 0, 1, 1);
    vert.anchor = GridBagConstraints.SOUTHEAST;
    add(blankPanel, gbl, vert, 2, 1, 1, 1);
    resize((imgwidth + 220), imgheight);
}
}

/* I don't know if I really need this; this might be necessary for */
/* regular applications only. */
public boolean handleEvent(Event evt) {
    if (evt.id == Event.WINDOW DESTROY) System.exit(0);
    return super.handleEvent(evt);
}

/**************************** action method ****************************/
/**************************** Deals with button clicks *********************/
public boolean action(Event evt, Object arg) {
    if (arg.equals("Submit Text")) {
        answered = true;
        // System.out.println("subtext answered = " + answered + ":\n");
        String youranswerstring = youranswer.getText();
        if (youranswerstring.equals(answerstring)) {
        }
    } else {
        if (arg.equals("Clear Text")) {
            clearButtonActionPerformed(null);
        }
        return super.action(evt, arg);
    }
}
sol.setText("Correct!");
    } else {
        sol.setText("Wrong!");
    }
}
else if (arg.equals("Show Answer")) {
    if (answered == true) {
        answer.setText(answerstring);
    } else {
        answer.setText("Submit an answer first");
    }
    // System.out.println("answered: "+ answered + "\n");
} else if (arg.equals("Clear Text")) {
    youranswer.setText("");
    answer.setText("" );
    sol.setText("" );
    answered = false;
} else return false;
return true;
} catch (Exception e) {
    System.out.println(e.getMessage());
    return false;
}

/******************************************************************************
***************************
NameCan.class ****************************
* An extension of Canvas which displays the image and bandwidth polygon *
******************************************************************************/
class NameCan extends Canvas {
    Image imagine;
    Polygon ipoly;
    Polygon opoly;
    int imgheight;
    int imgwidth;

    public NameCan(Image i, int w, int h, Polygon p, Polygon q) {
        imagine = i;
        ipoly = p;
        opoly = q;
        imgheight = h;
        imgwidth = w;
        resize(imgwidth, imgheight);
    }

    /** update method (overridden) ***************/
    /** Get rid of screen flicker ***************/
    public void update(Graphics g) {
        paint(g);
    }

    public void paint(Graphics g) {
        // Draw image and bandwidth polygon
    }

    // Public methods for updating answer and solution
}
/* Draw the image */
int imageWidth = imagine.getWidth(this);
int imageHeight = imagine.getHeight(this);
g.drawImage(imagine, 0, 0, this);

/* Draw reference polygon */
g.setColor(Color.red);
g.drawPolygon(ipoly);
g.drawPolygon(opoly);

E.3 Outline.java

// Outline.java

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.util.*;
import java.lang.*;

public class Outline extends Applet {

    /* Data used by main class */
    Image picture; /* The image used; parameter gotten from the HTML file */
    int correct_outline; /* Tells me to show "correct" or "incorrect" */
    Polygon innerpoly = new Polygon(); /* Inner part of bandwidth polygon */
    Polygon outerpoly = new Polygon(); /* Outer part of bandwidth polygon */

    private OutlineCan drawcanvas; /* The canvas where the cool stuff happens */
    private int imgheight; /* Image height */
    private int imgwidth; /* Image width */
    private String answerstring; /* The correct answer */
    private String dummyx; /* dummy variable for tokenizing x-coordinates */
    private String dummyy; /* dummy variable for tokenizing y-coordinates */
    private String name; /* The name of the image (ends in .jpg) */
    private String xinstring; /* x-coordinates of inner polygon */
    private String xoutstring; /* x-coordinates of outer polygon */
    private String yinstring; /* y-coordinates of inner polygon */
    private String youtstring; /* y-coordinates of outer polygon */
    private TextField answer; /* The field for "right" or "wrong" */
    private TextField rightorwrong; /* Where to show "correct" for outlines */
    private TextField sol; /* The field for the correct answer */
    private TextField youranswer; /* The field for the student's answer */

    private void add (Component c, GridBagLayout gbl, GridBagConstraints gbc, int x, int y, int w, int h) {
        gbc.gridx = x;
        gbc.gridy = y;
        gbc.gridwidth = w;
    }
}
gbc.gridheight = h;
gbl.setConstraints(c, gbc);
add(c);
}

/******************** init method ***********************************/
/******************** Main initialisation method *************************/
public void init() {

/* Get parameters from the html page */
answerstring = getParameter("featurename");
nname = getParameter("img");
xinstring = getParameter("innerx").trim();
yinstring = getParameter("innery").trim();
xoutstring = getParameter("outerx").trim();
youtstring = getParameter("outery").trim();
String dummyw = getParameter("iwidth").trim();
String dummyh = getParameter("iheight").trim();
imgwidth = Integer.parseInt(dummyw);
imgheight = Integer.parseInt(dummyh);

/* Tokenize the data for the bandwidth polygon coordinates */
StringTokenizer tix = new StringTokenizer(xinstring, ",");
StringTokenizer tiy = new StringTokenizer(yinstring, ",");
StringTokenizer tox = new StringTokenizer(xoutstring, ",");
StringTokenizer toy = new StringTokenizer(youtstring, ",");

int ptsin = tix.countTokens();
int ptsout = tox.countTokens();
int[] innerx = new int[ptsin];
int[] innery = new int[ptsin];
int[] outerx = new int[ptsout];
int[] outery = new int[ptsout];

for(int i=0; i<ptsin; i++) {
    dummyx = tix.nextToken();
    dummyy = tiy.nextToken();
    // System.out.println(dummyx + "," + dummyy + 
    innerx[i] = Integer.parseInt(dummyx);
    innery[i] = Integer.parseInt(dummyy);
}

for(int j=0; j<ptsout; j++) {
    dummyx = tox.nextToken();
    dummyy = toy.nextToken();
    // System.out.println(dummyx + "," + dummyy + 
    outerx[j] = Integer.parseInt(dummyx);
    outery[j] = Integer.parseInt(dummyy);
}

Polygon innerpoly = new Polygon(innerx, innery, ptsin); /* in band poly */
Polygon outerpoly = new Polygon(outerx, outery, ptsout); /* out band poly */

/* Setup grid */
GridBagLayout gbl = new GridBagLayout();
setLayout(gbl);

String pictureUrl = "images/" + name;

picture = getImage(getDocumentBase(), pictureUrl);
drawcanvas = new OutlineCan(picture, imgwidth, imgheight, innerx, innery,
    outerx, outery, innerpoly, outerpoly);
drawcanvas.resize(imgwidth, imgheight);
youranswer = new TextField("",40);
submitButton = new Button("Submit Text");
clearButton = new Button("Clear Text");
sol = new TextField("",15);
sol.setEditable(false);
showButton = new Button("Show Answer");
label = new Label("Answer:");
answer = new TextField(answerstring, 40);
answer.setEditable(false);
blankPanel = new Panel();
showOut = new Button("Show Outline");
rightOut = new Button("Check Outline");
clearOut = new Button("Clear Outline");
rightorwrong = new TextField("", 50);
rightorwrong.setEditable(false);

/* Horizontal layout for landscape-oriented pictures */
if (imgwidth > imgheight) {
    Panel horiz1 = new Panel();
    horiz1.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz1.add(youranswer);
    horiz1.add(submitButton);

    Panel horiz2 = new Panel();
    horiz2.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz2.add(label);
    horiz2.add(answer);

    Panel horiz3 = new Panel();
    horiz3.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz3.add(sol);

    Panel horiz4 = new Panel();
    horiz4.setLayout(new FlowLayout(FlowLayout.LEFT));
    horiz4.add(showButton);
    horiz4.add(clearButton);

    Panel horiz5 = new Panel();
    horiz5.setLayout(new FlowLayout(FlowLayout.CENTER));
    horiz5.add(rightOut);
    horiz5.add(showOut);
    horiz5.add(clearOut);

    Panel horiz6 = new Panel();
    horiz6.add(rightorwrong);

    Panel outPanel = new Panel();
    outPanel.resize(imgwidth,(imgheight+120));
}
outPanel.setLayout(new GridLayout(3, 1));
outPanel.add(horiz2);
outPanel.add(horiz5);
outPanel.add(horiz6);

GridBagConstraints horiz = new GridBagConstraints();
horiz.fill = GridBagConstraints.BOTH;
horiz.anchor = GridBagConstraints.NORTHWEST;
horiz.weightx = 0;
horiz.weighty = 0;
add(drawcanvas, gbl, horiz, 0, 0, 5, 6);

horiz.fill = GridBagConstraints.NONE;
horiz.anchor = GridBagConstraints.SOUTHWEST;
horiz.weighty = 100;
horiz.weightx = 100;
add(outPanel, gbl, horiz, 0, 6, 5, 3);

/* Vertical layout for portrait-oriented pictures */
else {
Panel vert1 = new Panel();
vert1.setLayout(new FlowLayout(FlowLayout.CENTER));
vert1.add(submitButton);
vert1.add(clearButton);

Panel vert2 = new Panel();
vert2.setLayout(new GridLayout(5, 1));
vert2.add(answer);
vert2.add(leftOut);
vert2.add(showOut);
vert2.add(clearOut);
vert2.add(rightorwrong);
vert2.resize(220, imgheight);

Panel vert3 = new Panel();
vert3.setLayout(new FlowLayout(FlowLayout.CENTER));
vert3.add(showOut);
vert3.add(clearOut);

/* Panel vertPanel = new Panel();
vertPanel.setLayout(new GridLayout(5, 1));
vertPanel.add(youranswer);
vertPanel.add(vert1);
vertPanel.add(sol);
vertPanel.add(showButton);
vertPanel.add(answer);
vertPanel.resize(220, imgheight);
*/

Panel vertPanel = new Panel();
vertPanel.setLayout(new GridLayout(6, 1));
vertPanel.add(label);
vertPanel.add(answer);
vertPanel.add(rightOut);
vertPanel.add(vert3);
vertPanel.add(rightorwrong);
vertPanel.resize(230, imgheight);

GridBagConstraints vert = new GridBagConstraints();
vert.fill = GridBagConstraints.BOTH;
vert.anchor = GridBagConstraints.NORTHWEST;
vert.weightx = 0;
vert.weighty = 0;
add(drawcanvas, gbl, 0, 0, 2, 2);
vert.fill = GridBagConstraints.NONE;
vert.anchor = GridBagConstraints.NORTHEAST;
vert.weighty = 100;
vert.weightx = 100;
add(vertPanel, gbl, 2, 0, 1, 1);
vert.anchor = GridBagConstraints.SOUTHEAST;
add(blankPanel, gbl, 2, 1, 1, 1);
resize((imgwidth+230),imgheight);
}

/**************************
** action method **********
***************************/
public boolean action(Event evt, Object arg) {

    // Shows the correct bandwidth outlines
    // There's some problem with the outline not showing up until the
    // mouse is moved into the canvas. Probably an update problem.
    if (arg.equals("Show Outline")) {
        boolean show_outline = drawcanvas.showCorrect();
        Graphics g = drawcanvas.getGraphics();
        repaint();
        if (!show_outline)
            rightorwrong.setText("Draw an outline first.");
    }
    else if (arg.equals("Check Outline")) {
        boolean correct_outline = drawcanvas.rightWrong();
        if (correct_outline == 1)
            rightorwrong.setText("You’re right!");
        // System.out.println("You’re right!");
        else if (correct_outline == 2)
            rightorwrong.setText("Wrong; press Clear and try again.");
        // System.out.println("Wrong; press Clear and try again");
        else
            rightorwrong.setText("Finish drawing the polygon.");
        // System.out.println("Finish drawing & then check.");
    }
    else if (arg.equals("Clear Outline")) {
        drawcanvas.clearvariables();
        rightorwrong.setT
else return super.action(evt, arg);
return true;
}
}

/******************* OutlineCan.class *******************
*****
This class is where the image is displayed on a canvas *****
***********************************************************/

class OutlineCan extends Canvas {
    final int MAXLINES = 30;  /* Max number of lines */
    final int TOLERANCE = 8;  /* How close before snapping to */
    boolean done = false;     /* Am I done drawing the polygon? */
    boolean rightout = false; /* Is the student polygon correct? */
    boolean showout = false;  /* Do we show the correct outline? */
    final double FRACTION_ALLOWED = .4;  /* Allowable fraction of vertices
          (that can be outside ref.
          polygons) */
    Image imagine;           /* Local name for image object */
    int beginx;              /* The x coord of the start point */
    int beginy;              /* The y coord of the start point */
    int currline = 0;        /* number of lines */
    int curx;
    int curry;
    int imgheight;
    int imgwidth;
    int[] inax;
    int[] inay;
    int[] outax;
    int[] outay;
    int[] student_x = new int[(MAXLINES + 1)]; /* x—coords of drawn polygon */
    int[] student_y = new int[(MAXLINES + 1)]; /* y—coords of drawn polygon */
    int[] xinner = new int[MAXLINES];         /* Reference Polygon outer & inner */
    int[] yinner = new int[MAXLINES];         /* vertices. Though I don't have */
    int[] xouter = new int[MAXLINES];         /* have to configure the polygons */
    int[] youter = new int[MAXLINES];         /* in the main class now. */
    Point vertices[] = new Point[MAXLINES + 1]; /* starting points */
    Point dummy[] = new Point[MAXLINES + 1];  /* starting points */
    Point anchor = null;                      /* start of current line */
    Point oldanchor = null;                   /* start of previous line */
    Point currentpoint = null;                /* current end of line */
    Point beginning = null;                   /* an alias for int = vertices[0] */
    Polygon ipoly;                           /* The inner and outer bandwith */
    Polygon opoly;                           /* polygon vertices */

    /************ OutlineCan Constructor Methods ************/
    /************ Set up the damn Constructor Method with ************
          the type of variables it can take! ************/

    public OutlineCan(Image i) {
        imagine = i;
    }

}
public OutlineCan(Image i, int inax, int inay, int outax, int outay) {
    imagine = i;
}

public OutlineCan(Image i, Polygon p, Polygon q) {
    imagine = i;
    ipoly = p;
    opoly = q;
}

public OutlineCan(Image i, int w, int h, int[] ix, int[] iy, int[] ox, int[] oy, Polygon p, Polygon q) {
    imagine = i;
    imgheight = h;
    imgwidth = w;
    inax = ix;
    inay = iy;
    outax = ox;
    outay = oy;
    ipoly = p;
    opoly = q;
    resize(imgwidth, imgheight);
}

/** Read the down-click of the mouse & deal with that data */

/************************** mouseDown method (overridden) **************************/
/************************** overrides standard mouseDown method **************************/
public boolean mouseDown(Event evt, int x, int y) {
    anchor = new Point(x, y);
    if (currline == 0) {
        beginning = anchor;  /* This point is the starting point */
        beginx = x;
        beginy = y;
    }
    if (done) {
        System.out.println("You're done drawing the polygon!");
    } else if (currline <= (MAXLINES - 1))
        addLine(x, y);
    // else if (currline == (MAXLINES - 1))
    //    addLine(vertices[0].x, vertices[0].y);
    else System.out.println("Too many lines");
    // System.out.println("currline=" + currline + " beginning= " + beginning);
    return true;
}

/************************** mouseMove method (overridden) **************************/
/************************** overrides standard mouseMove method **************************/
public boolean mouseMove(Event evt, int x, int y) {
    if (anchor == null)
        currentpoint = anchor;
    else
        currentpoint = anchor;
}
currentpoint = new Point(x,y);
repaint();
return true;
}

/*************************** addLine method ***************************
****** Keep track of the number of lines that've been drawn *******/
public void addLine(int x, int y) {
if (currline == (MAXLINES - 1)) {
    vertices[currline] = beginning;
    student_x[currline] = beginx;
    student_y[currline] = beginy;
    student_x[(currline+1)] = -1;
    student_y[(currline+1)] = -1;
    done = true;
}
else if (((currline > 0) && currline < (MAXLINES - 1)) &&
    (currentpoint.x > (beginning.x - TOLERANCE)) &&
    (currentpoint.x < (beginning.x + TOLERANCE)) &&
    (currentpoint.y > (beginning.y - TOLERANCE)) &&
    (currentpoint.y < (beginning.y + TOLERANCE)))) {
    vertices[currline] = beginning;
    student_x[currline] = beginx;
    student_y[currline] = beginy;
    student_x[(currline+1)] = -1;
    student_y[(currline+1)] = -1;
    // anchor = (beginning);
    done = true;
}
else {
    vertices[currline] = anchor;
    student_x[currline] = x;
    student_y[currline] = y;
}
// System.out.println("vertex[" + currline + "] = " + vertices[currline]);
currentpoint = vertices[currline];
currline++;
repaint();
}

/*************************** bandwidthInit method ***************************/
****** Set up the arrays with the bandwidth polygon vertices *******/
public void bandwidthInit() {
    System.arraycopy(inax, 0, xinner, 0, (inax.length));
    System.arraycopy(inay, 0, yinner, 0, (inay.length));
    System.arraycopy(outax, 0, xouter, 0, (outax.length));
    System.arraycopy(outay, 0, youter, 0, (outay.length));
    xinner[(inax.length)] = -1;
    yinner[(inay.length)] = -1;
    xouter[(outax.length)] = -1;
    youter[(outay.length)] = -1;
}

/*************************** showCorrect method ***************************/
**** Let the paint method know it's time to draw the correct outline ****/
public boolean showCorrect() {
    if (done) {
        showoutline=true;
        return true;
    }
    else
        return false;
}

/********************** rightWrong method **********************/
/****************** Press a button to show the user if they're right or not **********/
public int rightWrong() {
    /* Add the -1 to the arrays */
    bandwidthInit();

    if (done) {
        rightoutline = checkPolygon();
        if (rightoutline)
            return 1;
        else
            return 2;
    }
    else return 3;

    450

/********************** clearVariables method **********************/
/****************** Clears the variables so you can draw them all & try again *****/
public void clearVariables() {
    done = false;
    for (int j = 0; j < (MAXLINES + 1); j++) {
        vertices[j] = new Point(0,0);
        student_x[j] = 0;
        student_y[j] = 0;
    }
    showoutline = false;
    anchor = null;
    oldanchor = null;
    currentpoint = null;
    beginning = null;
    beginx = 0;
    beginy = 0;
    currline = 0;
}

/********************** update method (overridden) *******************/
/**************** Get rid of screen flicker *******************/
public void update(Graphics g) {
    paint(g);
}

/********************** paint method (overridden) *******************/
/**************** Draw the image, the existing & current lines, & maybe the answer *****/
public void paint(Graphics g) {
```java
int imageWidth = imagine.getWidth(this);
int imageHeight = imagine.getHeight(this);
g.drawImage(imagine, 0, 0, this);

/* Draw reference polygon if it's okay to do so */
if (showoutline) {
    g.setColor(Color.red);
    g.drawPolygon(ipoly);
    g.drawPolygon(opoly);
}

/* Draw existing lines */
for (int i=1; i < currline; i++) {
    g.setColor(Color.yellow);
    g.drawLine(vertices[i-1].x, vertices[i-1].y, vertices[i].x, vertices[i].y);
}

/* draw current line */
g.setColor(Color.blue);
// System.out.println("done = " + done + ", currentpoint = " + currentpoint + "\n");
if (done); else if (currline == MAXLINES); else if ((currentpoint != null) && (currline < MAXLINES)) {
    /*
     * This bit of code snaps the end of the line to the first point if
     * the end vertex is close enough. Grabbed from Pat's polyfuncs.c
     * code.
     */
    if (currentpoint.x > (beginning.x - TOLERANCE) &&
        currentpoint.x < (beginning.x + TOLERANCE) &&
        currentpoint.y > (beginning.y - TOLERANCE) &&
        currentpoint.y < (beginning.y + TOLERANCE))
        g.drawLine(anchor.x, anchor.y, beginning.x, beginning.y);
    else
        g.drawLine(anchor.x, anchor.y, currentpoint.x, currentpoint.y);
}

/**************************** checkPolygon method ****************************/
/****** Checks to see if polygon resembles answer (bandwidth) ******/
public boolean checkPolygon() {
    int stud_area; /* Area of student polygon */
    int num_wrong_vert = 0; /* Number of vertices outside of bandwidth */
    int j; /* Dummy counter */
    int foo;
    int bar;
    double divider;

    /* Calculate area of student polygon. */
    stud_area = polygonArea(student_x, student_y);
    // for (int i=0; i<student_x.length; i++) {
    //    System.out.println("\" + student x[i] + \", \" + student_y[i] + \"\n");
    // }
    System.out.println("stud_area = " + stud_area + "\n");
    foo = polygonArea(xinner, yinner);
    bar = polygonArea(xouter, youter);
```
for (int m=0; m<xinner.length; m++) {
    System.out.println("(" + xinner[m] + ", " + yinner[m] + ")");
}
for (int n=0; n<xouter.length; n++) {
    System.out.println("(" + xouter[n] + ", " + youter[n] + ")");
}
System.out.println("ipolyarea = " + foo + " opolyarea = " + bar + ");
/*
 * To avoid decimal errors, the areas are represented as twice the number
 * of pixels.
*/
/*
 * Polygon vertex is wrong if student area does not have a value that
 * lies between inner and outer polygons.
*/
if ((stud_area < polygonArea(xinner, yinner)) ||
    (stud_area > polygonArea(xouter, youter))) {
    /* Polygon fails from area criteria. */
    System.out.print("check_polygon: polygon is incorrect: wrong size\n");
    return false;
}
/*
 * Since area meets requirement: Check whether student's polygon is
 * inside bandwidth polygon.
*/
j=1;  /* Counter variable. */
/* Checks all student vertices except the first because it's the last one. */
while (studentx[j] != -1) {
    if (ipoly.inside(studentx[j], studenty[j])) {
        ++num_wrong_vert;  /* Wrong: Vertex inside inner bandwidth. */
        System.out.println("bad in; num_wrong_vert = " + num_wrong_vert + ");
    }
    /* Next, for each vertex, if it's outside the outer polygon, then
     * it's wrong.
     */
    else if (!opoly.inside(studentx[j], studenty[j])) {
        ++num_wrong_vert;  /* Wrong: Vertex outside outer bandwidth. */
        System.out.println("bad out; num_wrong_vert = " + num_wrong_vert + ");
    }
    ++j;  /* Increment index -> Check next student vertex. */
}
/* Calculate the fraction of vertices outside the bandwidth */
double divider3 = (double) num_wrong_vert / (j-1);
System.out.println("divider3 = " + divider3 + "; j = " + j + "; num_wrong_vert =" + num_wrong_vert + ");
if (((double) num_wrong_vert / (j-1)) >= FRACTION_ALLOWED) {
/* Polygon fails from vertex criteria. */
// System.out.print("check_polygon: polygon is incorrect: bad vertices\n");
System.out.print("Percent wrong: " + (double) num_wrong_vert / (j-1) + "\n");
return false;
}
else {
    /* Polygon is correct */
// System.out.print("Percent wrong: " + (double) num_wrong_vert / (j-1) + "\n");
    System.out.print("check_polygon: polygon is correct\n");
    return true;
}

/*********************** findArea method ******************************/
/*********************** Finds the area of the polygons (template) *************/
public static int findArea(int xl, int yl, int x2, int y2, int x3, int y3) {
    int findarea =
        (-xl*y2) + (xl*y3) + (x2*yl) - (x2*y3) - (x3*yl) + (x3*y2);
    // if (findarea < 0)
    //     findarea = -1 * findarea;
    return findarea;
}

/*********************** polygonArea method ******************************/
/*********************** Returns twice the number of pixels circumscribed by polygon ******/
public static int polygonArea(int[] x, int[] y) {
    int poly_area = 0; /* Initialization of Area */
    int j = 0; /* Dummy Counter */

    /* If these are less than 3 vertices, the area is zero */
    if (x[0] == (-1) || x[1] == (-1)) 630
        return (0);
    while (x[j+2] != (-1)) {
        poly_area += findArea(x[0], y[0], x[j+1], y[j+1], x[j+2], y[j+2]);
        ++j;
    }
    if (poly_area > 0)
        return (poly_area);
    else
        return (-poly_area);
}
Appendix F

Workbook Page Generation Script Code

#!/usr/athena/bin/perl
#
# This file written by Jennie Hango, February 1997
# Last modified 31 July 1997 by Jennie Hango
# The first part of the file will have the data that all of the
# script-created files will need, like the name of the data file, the
# chapter name, etc.
#
# These are some initialised variables used for loops
#
$count = 1;
$foo = 1;

# This chunk of code reads the name of the data file. This will need
# to be input by the user of the program.

print "Please enter the name of the data file to process and press <return>\n"
chop($datafile = <STDIN>);

# This is the actual reading of the data for the generated files.
# This line opens the data file for reading.

open(DB, "$datafile") || die "$datafile: $!
";
INITIALISE: while (<DB>){
    next INITIALISE if ($_ =~ /^\s*#/);
    last INITIALISE if ($foo > 5);

    # $foo == 0 gets rid of the blank line that separates the data for
    # each file.
    if ($foo==0) {
        $foo++;
    }
    elsif ($foo == 1) {
        chop($_);
        $chapter = $_;
        $foo++;
    }
    elsif ($foo == 2) {
        chop($_);
        $exercise = $_;
        $foo++;
    }
    elsif ($foo == 3) {
        chop($_);
        $applet = $_;
        $foo++;
    }
    elsif ($foo == 4) {
        chop($_);
        $addhoriz = $_;
        $foo++;
    }
    else {
        chop($_);
        $addvert = $_;
        $foo++;
    }

    # $chapter = chapter name
    # $capchapter = capitalised chapter name
    # $cha = first 3 letters of uncapitalised chapter name
    # $htmlfile = name of workbook page
    # $prevfile = name of previous file
    # $indexfile = name of chapter index file
    # $exercise = type of exercise: both, name, or outline
    # $applet = applet name
    #
    $capchapter = $chapter;
    substr($capchapter,0,1) =~ tr/a-z/A-Z/;
    $cha = $chapter;
$cha =~ s/(\().(\).*\2\3)/\1\2\3/;
substr($cha,0,1) =~ tr/A-Z/a-z/;
$exercise =~ tr/A-Z/a-z/;
substr($applet,0,1) =~ tr/a-z/A-Z/;

$htmlfile = $cha . $exercise . $count . ".html";
$nextfile = $cha . $exercise . ($count + 1) . ".html";
$indexfile = $cha . "index.html";

# Which $prevfile and $extype to use depends on the workbook exercise type

if ($exercise eq "both")
  {$prevfile = $cha . "exboth.html";
   $extype = "Name and Outline";
  }
elsif ($exercise eq "name")
  {$prevfile = $cha . "exname.html";
   $extype = "Name Only";
  }
elsif ($exercise eq "out")
  {$prevfile = $cha . "exout.html";
   $extype = "Outline Only";
  }
else
  {die "Wrong exercise type: $! \n";
  }

# This bit of code gets the first file's data before entering the loop
# re-initialise $foo for the next bit of data

$foo = 1;

FIRSTREAD: while (<DB>){
  next FIRSTREAD if ($_. =~ /^\s*\#/) 100
  last FIRSTREAD if ($foo > 10);
  if ($foo==0)
    {$foo++;
     }
  elsif ($foo == 1)
    {chop($_);
     $placeholder = $_ . ":";
     $foo++;n
    }
  elsif ($foo == 2)
    {chop($_);
     $placeholder = $placeholder . $_ . ":";
     $foo++;n
    }
  elsif ($foo == 3)
    {chop($_);
     $placeholder = $placeholder . $_ . ":";
     $foo++;n
    }
  elsif ($foo == 4)
    {chop($_);
     $placeholder = $placeholder . ".";
     $foo++;n
    }
  elsif ($foo == 5)
    {chop($_);
     $placeholder = $placeholder . ".";
     $foo++;n
    }
  elsif ($foo == 6)
    {chop($_);
     $placeholder = $placeholder . ".";
     $foo++;n
    }
  else
    {die "Wrong exercise type: $! \n";
     }
  }
elsif ($foo == 4) {
    chop($_);
    $placeholder = $placeholder . $_. "":";
    $foo++;
}
elsif ($foo == 5) {
    chop($_);
    $placeholder = $placeholder . $_. "":";
    $foo++;
}
elsif ($foo == 6) {
    chop($_);
    $placeholder = $placeholder . $_. "":";
    $foo++;
}
elsif ($foo == 7) {
    chop($_);
    $placeholder = $placeholder . $_. "":";
    $foo++;
}
elsif ($foo == 8) {
    chop($_);
    $placeholder = $placeholder . $_. "":";
    $foo++;
}
elsif ($foo == 9) {
    chop($_);
    $placeholder = $placeholder . $_. "":";
    $foo++;
}
else {
    chop($_);
    $placeholder = $placeholder . $_;  
    $dataline = $placeholder;
    $foo++;
}
}

# The first file's data is now in a variable. Now we enter the main
# loop so that we can get all of the rest of the files' data.
#
# Once again, we need to reset $foo
#

$foo = 1;

READDATA: while (<DB>){
    next READDATA if ($_ =~ /\*\*\#/);
    if ($foo == 0) {
        $foo++;
    }
    elsif ($foo == 1) {
        chop($_);
$placeholder = $_. "::";
$foo++;
}
elsif ($foo == 2) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 3) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 4) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 5) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 6) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 7) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 8) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
elsif ($foo == 9) {
    chop($);
    $placeholder = $placeholder . $_. "::";
    $foo++;
}
else {
    chop($);
    $placeholder = $placeholder . $;
    $foo++;

# Now the fun begins....
#
=openhtml;
=writehtml;
#
# Now the fun ends....
#
$dataline = $placeholder;
$foo = 0;
$count++;
$htmlfile = $cha . $exercise . $count . "html";
$prevfile = $cha . $exercise . ($count - 1) . "html";
$nextfile = $cha . $exercise . ($count + 1) . "html";

close(WKBKPG);
}

close(WKBKPG);

# These next few lines of code write the last file of the series, because
# it doesn't follow the naming convention of the non-first files.

&openhtml;

# Which $nextfile to use depends on the type of workbook exercise it is.
if ($exercise eq "both")
{$nextfile = $cha . "exname.html";}
elsif ($exercise eq "name")
{$nextfile = $cha . "exout.html";}
else
{$nextfile = "./" . $cha . "index.html";}

&writehtml;

close(WKBKPG);
close(DB);

print "Processing completed. $count files were generated.\n";

#
# SUBROUTINES
#
# Photographic references
#
###
### &references
###

sub references {
  if ($reference eq "Ward") {
    $refname = "Original picture from the Ward Collection";
  }
  elsif ($reference eq "Shelton") {
    $refname = "Earth Science Slides by John Shelton";
  }
  elsif ($reference eq "Einstein") {
    $refname = "Original picture by Herbert H. Einstein,"}
elif ($reference eq "Allen") {
    $refname = "Allen et al., NAGT Death Valley Slides";
}
elsif ($reference eq "Hango") {
    $refname = "Original picture by Jennie Hango, Massachusetts Institute of Technology";
} else {
    $refname = "Unknown photographic source";
}

# These two subroutines deal with the generation of the .html files.
# openhtml opens the file to write to, and writehtml actually writes # the file.
#
### openhtml
###
sub openhtml {
    open(WKBKPG, ">$htmlfile") || die "$htmlfile: $!
";
    $ = $dataline;
    ($image,$width,$height,$name,$reference,$textbook,$innerx,$innery,$outerx,$outery)=split(":");
    if ($width < $height) {
        $horiz = $width + $addhoriz;
        $vert = $height;
    } else {
        $horiz = $width;
        $vert = $height + $addvert;
    }
&references;
}

### writehtml
###
sub writehtml {
    print WKBKPG (<<EOT);
    <html>
    <head>
    <title>$capchapter Workbook</title>
    </head>
    <body background="../common/bkg.gif">
    <h1>$capchapter Workbook - $extype #$count</h1>

Bibliography


