Development of Interactive and Real-Time Educational Software for Mechanical Principles

by

Jochen Schlingloff

Submitted to the Department of Civil and Environmental Engineering in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Civil and Environmental Engineering
At the
Massachusetts Institute of Technology

June 2001

Signature of author: _____

Jochen Schlingloff
Department of Civil and Environmental Engineering
May 11, 2001

Certified by: _____

Kevin Amaratunga
Associate Professor, Department of Civil and Environmental Engineering
Thesis Supervisor

Accepted by: _____

Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

Development of Interactive and Real-Time Educational Software for Mechanical Principles

by

Jochen Schlingloff

Submitted to the Department of Civil and Environmental Engineering on May 11, 2001 in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Civil and Environmental Engineering

Abstract

This document is based on the work that was part of the 'Flagpole project', a research project in the Department of Civil and Environmental Engineering at the Massachusetts Institute of Technology (MIT). Its focus is on the development of interactive and real-time educational software that could be used as a supplementary tool in mechanics education.

In the beginning, the document contains an overview of the Flagpole project and the problem statement that was part of this project, i.e. to develop software that improves the learning experience of students through hands-on studies with cutting-edge teaching tools.

An overview of the Java applets developed during that project is provided as well as an in-depth explanation of one example applet. In addition, one chapter discusses different approaches to the software development cycle.

Finally, the achievements of the Flagpole project and possible future research options are summarized, and the code of the example software that is explained in more detail is appended.

Thesis Supervisor: Kevin Amaratunga
Title: Assistant Professor of Civil and Environmental Engineering

Acknowledgements

I would like to thank my advisors, Professor Kevin Amaratunga for his help and guidance throughout the flagpole project and this thesis, and Professor George Kocur for his advice throughout the course of this year.

Furthermore, I would like to give my thanks to my fellow students in the Flagpole project and the Master of Engineering Program, especially the guys who taught me the subtleties of the English language and never ceased to make fun of me when I was a stupid foreigner. You have been a very important part of this year for me and I really hope to see you again some day!

I also don't want to forget my friends back in Germany, who showed me that true friendship also lives over long distances.

But most of all, I want to thank my family and my girlfriend Andrea for their endless love and support. I am very grateful for all the opportunities you gave me and I will always remember it. Words cannot express my gratitude, but I hope that my actions will do. Tausend Dank und auf ein fröhliches Wiedersehen in Deutschland!

Table of Content

Table of Figures

1 Introduction

1.1 Motivation

We are living in a world that is characterized by technological innovations and constantly increasing amounts of available data. All these data are meaningless by themselves and have to be gathered, organized, and analyzed in order to become useful information. This can be achieved with the help of new high-technology products, which are developed faster and faster as their life cycle becomes shorter and shorter.

These new technologies do not aim to replace more traditional sciences and technologies but rather to expand their capabilities. Biotechnology offers new ways of medical examination and treatment. New planes and trains speed up traveling and make it easy to access regions previously considered inaccessible. The Internet, cellular phones, laptops, organizers and innumerable other devices facilitate communication and fast transfer of information all around the world.

The impact these developments have had on the profession of civil engineering is enormous. Sophisticated software and increasing computational capabilities provide the possibility to calculate and erect very complex structures. A single computer can execute an analysis that would have a required a month's work from a number of engineers twenty years ago in a few seconds. But a computer is only a machine that does what it is told to do. Therefore, it is very important to understand the engineering tasks and problems that are part of the building process to avoid the infamous 'garbage in – garbage out' result of a computer-supported structural analysis.

The basic principles of civil engineering remain unchallenged in their importance and it is the task of the universities to combine the basic engineering expertise with the knowledge of newly emerging technologies in a student's education. The universities have to face this challenge to provide their students with a widened education that prepares them well for today's diverse civil engineering tasks.

This document is going to explain one of many possible ways to improve education, the development of software products for educational purposes, in order to prepare today's students for the tasks of tomorrow.

1.2 The Flagpole Project

The Flagpole project is a research project at the Massachusetts Institute of Technology (MIT). It was started in the fall semester of the year 2000 as a project for the students in the Master of Engineering Program in the Department of Civil and Environmental Engineering. The Microsoft Education Fund sponsors the project.

The Flagpole project team consists of two Master of Science students and eight Master of Engineering students under the leadership of Professor K. Amaratunga. While the Master of Engineering students are only able to participate for one year because of the program's duration, the management by Professor Amaratunga and the involvement of the Master of Science students guarantee the continuation of the project.

The basic goals of the Flagpole project are to create a decision-support system for real-world structural systems as well as to improve structural education through the use of new technologies. This document focuses mainly on the educational aspect of the Flagpole project, only a brief overview of the decision-support system is provided in this chapter. A flagpole on the MIT campus was chosen as an example structure to develop an initial decision-support system that ideally would be expandable to larger real-world structures in the future.

The team's effort to create a structural decision-support system that is able to react to changing external loads and conditions can be divided into three main areas: hardware, software, and visualization. The figure below shows these three areas and their characteristic parts.

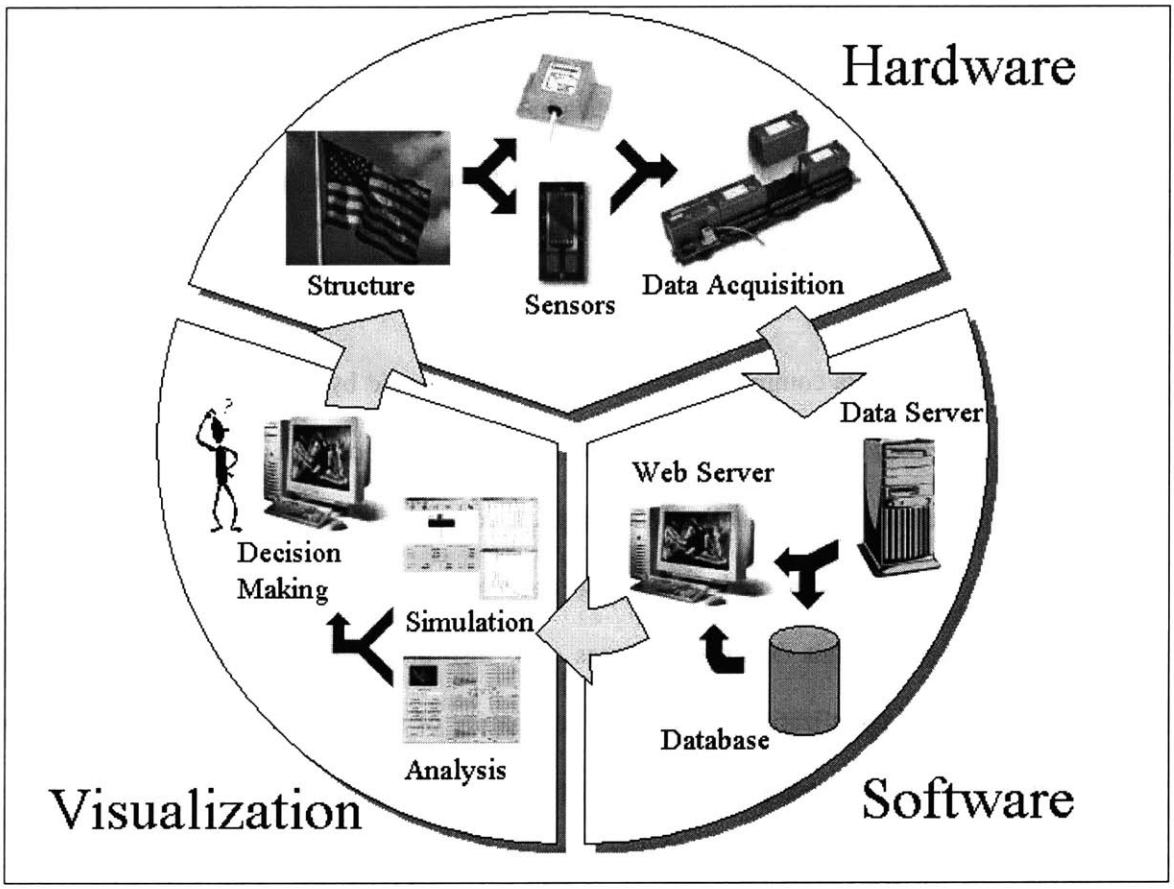

Figure 1-1:The Flagpole Project parts for a decision-support system

Hardware: The hardware section contains all the instrumentation that is necessary to collect and prepare the data from the flagpole as well as the flagpole structure itself. Although the flagpole has been replaced with an aluminum prototype to test the hardware equipment until now, the flagpole itself will be equipped with the sensors in the near future. The goal is to monitor the loads and the structural reaction through the data coming from the sensors. Different types of sensors are used to capture the characteristic loads and system responses. An anemometer measures the wind speed in the environment of the flagpole, the main source of loading in this particular case. Strain gauges measure the structural strains as an indicator of the systems behavior. To monitor the dynamic system responses, accelerometers are attached to the structure.

Software: The software section takes care of collecting and storing the data delivered from the hardware components. It is possible to get the data in real-time as it is coming from the sensors as well as to retrieve old data from a database. Server software is used to publish this data and to make it available for further analysis.

Visualization: The raw data coming from the sensors and collected by the software section has to be edited and visualized in order to become a meaningful analysis of the events at the structure's location. This is the task of the visualization section. It contains software for a structural analysis as well as software to analyze and display the collected data in a well-ordered format. The educational software that this document focuses on is also part of this section of the Flagpole project. Although the visualization section consists mainly of software, it is separated from the section called 'software' to stress the fact that it focuses on the data analysis in order to lay the basis for decision support. In a future scenario, this section could be used not only to study the

system behavior but also to influence flexible system parameters, e.g. the damping, in order to react to changing circumstances.

An overview of the Flagpole project and the software that is the topic of this document can be found on the Internet under http://flagpole.mit.edu.

1.3 Problem Statement

University education today is based mainly on lectures, assignments, and exams. For some courses, the material is put on the Internet, but that is not a standard today. The Flagpole project wants to create a virtual laboratory that could be used in education to provide hands-on experience for student learning. By giving them the opportunity to apply the theoretical knowledge gained from lectures and assignments to a real-world system, their preparation for professional practice would be improved.

Creating educational software poses the question of how people learn most effectively. For this reason, constant feedback and interaction with the users is an indispensable part of the Flagpole project to constantly improve the educational software. Therefore, it is one goal to make professors use our software in their classes in order to build a base of users that could be asked to provide their comments and proposals after using the software. However, since we did not have access to a large number of customers that our software is aimed at, we had to try to put ourselves in the position of a learning student during the initial software development phase.

The educational software of the Flagpole project does not claim to replace the more 'traditional' parts of a structural education, but it wants to enliven them by using cutting-edge technology.

Thus a basic familiarity with the corresponding mechanical concepts can be presupposed for our users. The advantage of using the software instead of doing it by hand is the reduction of execution time and the opportunity to study a large number of different system settings in a short period of time. However, the software does not relieve a student of the need to have at least a basic idea of the underlying mechanical concepts, otherwise the software will not be able to deepen his understanding.

A distinctive feature of the Flagpole project is the opportunity to study structural behavior real-time. A lot of existing software deals with archived data or delayed data, but dealing with data in real time is more difficult to do. As a benefit, dealing with real-time data allows the studying of dynamic processes as they occur, which makes it more exciting because of the opportunity to compare what one sees with the expectation one gets from applying the software.

There are numerous possible ways to attack this project. Some of them along with the one that the Flagpole team has chosen to go are explained in this document.

2 Software Development Process

2.1 The Software Development Process

The software development process can be defined as a framework for the tasks that are required to build high-quality software. Like any engineering approach, software development must rest on a commitment to quality. It is important to go through a series of predictable steps, a road map that helps to create a timely, high-quality result. Software developers adapt the process to their needs and then follow it. The key process areas form the basis for control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are achieved, quality is ensured, and change is properly managed. In addition, the people who are going to be the users of the final product play a role in the development process. It is important to follow a process because it provides stability, control, and organization to an activity that can become chaotic if it is left uncontrolled.

There are many different approaches to this process. This chapter introduces the most common software development models and their structures. It outlines their advantages and disadvantages as well as their appropriateness for certain software projects and the Flagpole project, respectively.

No matter which model one chooses to follow, it is important to analyze its suitability to the undertaken project. Software projects always contain a high risk of failing to deliver a product with high quality in time. The selection of an appropriate development model can help to reduce this risk, although it is not a guarantee for success on its own. Choosing the right approach is not

an easy task and requires some experience. However, once the project is finished, the quality, timeliness, and long-term viability of the software product that has been built are going to indicate the efficacy of the process that one had chosen to use.

Regardless of application area, project size, or complexity, the work associated with software development can be categorized into three generic phases:

The *definition phase* focuses on the question what is to be done. During this phase, the software developer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified.

The *development phase* focuses on the question of how it is to be done. During this phase, it has to be identified how the data are to be structured, how function is to be implemented within a software architecture, how interfaces are to be characterized, and how testing will be performed.

Finally, the *support phase* focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. This phase reapplies the steps undertaken during the definition and development phases in the context of existing software.

2.2 Software Process Models

2.2.1 The Linear Sequential Model

Also known as the *classic life cycle* or the *waterfall model*, the *linear sequential model* suggests a systematic, sequential approach to software development. Modeled after a conventional engineering cycle and beginning at the system level, it progresses through a series of activities that are described in the following paragraphs.

System/information engineering and modeling: Software is always part of a larger system. Therefore, work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. This view of software as a part of a larger system is especially important when software must interact with other system components such as hardware, people, and databases.

Software requirement analysis: During this phase, the requirements gathering process is intensified and focused particularly on software. The software developer must get an idea of the required function, behavior, performance and interface of the software. Requirements for both the system and the software are documented and reviewed with the customer.

Design: This is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representation, and procedural detail. Requirements are translated into a representation of the software that can be assessed for quality before coding begins. The design has to be documented thoroughly.

Code generation: The task that has to be performed now is the translation of the design into a machine-readable form. It is strongly dependent on the quality of the design. If the design has been performed well in a detailed manner, code generation can be done mechanistically.

Testing: After the code generation, testing begins with a focus on the logical internals and the functional externals of the software. It is important to ensure that all statements have been tested to uncover errors and that defined input will produce actual results that agree with required results.

Support: It is highly likely that software will undergo change after it has been delivered to the customer. The reasons for that may be encountering of errors, a need of adaptation to accommodate changes in the external environment (new operating systems, new peripheral devices, etc.) and customers requiring functional or performance enhancements. Rather than creating a new program, software support/maintenance reapplies each of the preceding phases to an existing program.

The linear sequential model is today's most widely used paradigm for software development. It provides a template into which methods for analysis, design, coding, testing, and support can be placed. Nevertheless, it also has its downsides. Since real projects rarely follow the sequential flow that the model proposes, changes can cause confusion as the project team proceeds. This model has its difficulties with the accommodation of the natural uncertainty that exists at the beginning of many projects, because it is often difficult for customers to state all requirements explicitly. In addition to that, the use of this model might lead to so-called 'blocking states' in

which some team members have to wait for others to fulfill their depending tasks. However, the linear sequential model has a definite and important place in software development work.

2.2.2 The Prototyping Model

The *prototyping model* may offer the best approach to developing software if a customer defines a general set of objectives but does not identify detailed input, processing, or output requirements. It may also be suitable in cases where developers are unsure about the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. Developers and customers start by defining the overall objectives for the software, identify whatever requirements are known, and outline areas of the project where further definition will be necessary. The following 'quick design' emphasizes the representation of those aspects that will be visible to the user and leads to the construction of a prototype. The customer according to his objectives evaluates this prototype and both customer and developer use it to refine requirements for the software to be developed. After that, the prototype is tuned to satisfy the needs of the customer. If a working prototype is built, the developer attempts to use existing program fragments or applies tools that enable working programs to be generated quickly. The advantages of this paradigm are that users get a feel for the actual system very early and that developers have the opportunity to build something immediately. The question that remains is what to do with the prototype. Here lies the danger of the prototyping model since customers might be trying to force the software developers to complete 'a few fixes' to make the prototype a working product after seeing what appears to be a working version of the software, unaware that in the rush to get it working no one has considered overall quality or long-term maintainability. The key to success is making both customers and developers agree in the beginning that the prototype is built to serve

as a tool for defining requirements and is then at least partially discarded. Although problems can occur, prototyping can be an effective paradigm for software development.

2.2.3 The RAD Model

An incremental software development process model that focuses on extremely short development cycles is the *rapid application development (RAD)* model. It is a 'high-speed' adaptation of the linear sequential model and uses component-based construction to achieve rapid development. The RAD process enables developers to build a working software system within short time periods, if the project scope is constrained and the requirements are well understood. It encompasses the following phases:

Business modeling: The initial phase deals with the modeling of the information flow among business functions. Its goal is to provide answers to the questions of what information drives the business, what information is generated, who generates it, where does the information go, and who processes it.

Data modeling: The previously defined information flow is refined into a set of data objects that are needed to support the business. The attributes of each object are identified and the relationships between these objects are established.

Process modeling: The data objects are transformed to achieve the information flow necessary to implement a business function. Additionally, processing descriptions are produced to add, modify, delete, or retrieve a data object.

Application generation: RAD prefers to reuse existing program components (when possible) and to create new reusable components (when necessary) rather than using conventional programming languages. It works with automated tools to facilitate the development of software.

Testing and turnover: Overall testing time is reduced because many of the components are reused and have already been tested. However, it is still necessary to test the new components and all interfaces.

RAD is an appropriate software development paradigm if it is possible to modularize a business application in a way that enables each major function to be completed in less than three months. In this case, separate RAD teams can address each major function and finally integrate them to form a whole. But RAD has also its disadvantages. First of all, it requires sufficient human resources to create the RAD teams and a commitment from developers and customers to the rapid-fire activities necessary for completing a system in an abbreviated time frame. Although it may be especially tempting to use RAD for projects that involve the heavy use of new technology in order to go out to the market early, it is not an appropriate paradigm in such cases because technical risks are too high.

2.2.4 The Incremental Model

The *incremental model* is part of a larger group of evolutionary software process models, which emerged from the recognition that software evolves over a period of time, like all complex systems. Nowadays, a straight path to an end product is often unrealistic, because business and product requirements frequently change as development proceeds and tight market deadlines make the timely completion of a comprehensive software product impossible. In such cases,

software developers need a paradigm that accounts for the evolution of the product. All evolutionary models are iterative and enable software developers to produce increasingly more complete versions of the software over time.

The incremental model offers a combination of the repetitive application of elements known from the linear sequential model with the iterative approach of prototyping. It delivers software in increments. Each of these small but usable pieces builds on those that have already been delivered. The first increment is often a core product that reflects the basic requirements without delivering supplementary features. After that, a plan for the next increment is developed according to the results of the customer's use and evaluation of this first increment. This process is repeated following the delivery of each increment, modifying the core product and adding additional features and functionality until the complete product is produced.

The advantage of the incremental model is that early increments are stripped down versions of the final product, which provide capabilities that serve the user and also provide a platform for evaluation. It is especially useful when a difficult deadline has been established that cannot be changed, because this model allows the implementation of early increments with fewer people and the addition of personnel when the project proceeds.

2.2.5 The Spiral Model

The *spiral model* is another evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development due to software building in a series of incremental releases.

A spiral model is divided into three to six framework activities, the so-called 'task regions', which usually contain customer communication, planning, risk analysis, engineering, construction and release, and customer evaluation. A task set is associated with each task region, whose work tasks are adapted to the size and characteristics of the project to be undertaken. As this evolutionary process begins, the software development team moves around the spiral, starting at the center. The first circuit around the spiral might result in the development of a product specification. Subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback deduced from customer evaluation. The project management also has the opportunity to fit the planned number of iterations to complete the software.

The spiral model is different from classical process models in the fact that it does not end when the software is delivered. It can be adapted to apply throughout the whole life cycle of a software product. This paradigm provides a realistic approach to the development of large-scale software. Because of the iterative process, the developer and the customer better understand and react to risks at each evolutionary level. Developers are enabled to apply the prototyping approach at any desired stage during the evolution of the product. The spiral model should significantly reduce project risks, if properly applied, because it demands a direct consideration of technical risks at all stages of the project. Nevertheless, this realistic and thorough process model has its drawbacks, too. First of all, it may be problematic to convince customers that this approach is controllable and does not go out of hand. It also demands considerable risk assessment expertise and relies on that for success. Finally, the model has not been used as widely as the more classical

software development process models and it might take some years until its efficacy can be determined with certainty.

2.2.6 The Formal Methods Model

The *formal methods model* encompasses a set of activities that leads to formal mathematical specifications of software. It applies a rigorous, mathematical notation to specify, develop, and verify a computer-based system with the support of formal methods.

Using the formal methods model for software development provides a mechanism for eliminating a number of problems that are difficult to overcome when using other paradigms. Through the application of mathematical analysis, it is easier to discover and correct ambiguity, incompleteness, and inconsistency. The formal methods model offers the promise of defect-free software, yet it is doubtful that it will become a mainstream approach, due to the following concerns: it is currently quite expensive and time-consuming to develop formal models; a lot of extensive training is required since not many software developers already have the necessary background to apply formal methods; finally, it is difficult to communicate with technically unsophisticated customers using this model.

The formal methods model will likely gain popularity among software developers who must build safety-critical software, e.g. for aircraft avionics or medical devices, and among those who would suffer severe economic hardship should software errors occur.

2.2.7 Fourth Generation Techniques

The term *fourth generation techniques (4GT)* represents a broad array of software tools that have one thing in common: each of them enables the software developer to specify some characteristics of the software at a high level. The tool then automatically generates source code based on the developer's specification. The 4GT paradigm for software development concentrates on the ability to specify software using specialized language forms or a graphic notation that describes the problem to be solved in terms that the customer can understand. Usually, most of the following features are contained in a software development environment that supports the 4GT paradigm: nonprocedural languages for database queries, report generation, data manipulation, screen interaction, and code generation; high-level graphics capabilities; spreadsheet capability and automated generation of HTML and similar languages used for Web-site creation.

The first phase of 4GT is requirements gathering. It is impossible to translate the customer's description of the requirements directly into a prototype hence the customer/developer dialog remains a very important part of the 4GT approach. In the next step, larger systems require a design strategy for the system, while small systems can be done without that. After this phase, the project moves to implementation using a nonprocedural fourth generation language (4GL) or a model composed of a network of graphical icons. Automatic code generation is achieved through the means of implementation using a 4GL that enables the software developer to represent results in a suitable way. Ultimately, in order to transform a 4GT implementation into a product, thorough testing has to be conducted, meaningful documentation has to be developed, and all remaining solution integration activities required in other software development paradigms have to be performed.

The greatest advantages of 4GT are the reduction of software development time and the improved productivity of people who build software. On the other hand, one might claim that 4GT tools are not that much easier to use than programming languages and that the resulting source code produced by such tools is inefficient.

Today, 4GT environments have been extended to address most software application categories. However, it is important to remember that even with the use of 4GT, one has to emphasize solid software development by doing analysis, design, and testing. 4GT are already an important part of software development and may become the dominant paradigm in the future.

2.2.8 Extreme Programming

Extreme programming (XP) is a new lightweight methodology for small-to-medium sized teams developing software in the face of vague or rapidly changing requirements. It is called 'extreme' programming because it takes commonsense principles and traditional practices to extreme levels, e.g. constant code review, perpetual testing, highest possible level of simplicity, continuous integration, and shortest feasible iterations. This paradigm challenges many conventional tenets, including the long-held assumption that the cost of changing a piece of software necessarily rises dramatically over the course of time.

XP takes an incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project. Short development cycles provide early, concrete, and continuing feedback. This model relies on automated tests written by programmers and customers to monitor the progress of development, to allow the system to evolve, and to

catch defects early. It uses oral communication, tests, and well-documented source code to communicate system structure and intent. It is furthermore distinguished from other process models by its ability to flexibly schedule the implementation of functionality, responding to changing customer needs. The design process that follows this model is supposed to last as long the system lasts.

The development cycle of XP can be summed up in a few characteristics. First of all, pairs of programmers program together. The development is driven by tests. The work is not done until all tests run perfectly. When all tests run and the developers cannot think of any more tests that might break, they are done adding functionality. Making test cases run is not the only task of the programming pairs. They are also expected to add value to the analysis, design, and implementation of the system. The integration of the software immediately follows development, including the integration testing.

XP acknowledges risk as the basic problem of software development and claims to address these risks before they are threatening the whole project. The following paragraphs explain the way in which XP addresses software development risks:

Schedule slips: The scope of any slip is limited from the very beginning, since XP calls for short release cycles that are not longer than a few months. Within a release, XP uses one- to four-week iterations of customer-requested features for fine-grained feedback about the project progress. One- to three-day tasks are planned within an iteration thus enabling developers to solve problems even during iteration. Finally, XP requires implementing the highest-priority features first, so any feature that slips past the release date will be of lower value for the customer.

Project canceled: The customer is asked to specify the smallest thinkable release that still makes sense from the business perspective, thereby the number of things that can possibly go wrong is minimized before going into production.

System goes sour: XP keeps the system in prime condition at all times with the help of a comprehensive suite of tests, which are run and re-run after every change to ensure a quality baseline.

Defect rate: The defect rate is reduced via tests that are written both from a programmer's point of view and from a customer's point of view.

Business misunderstood: XP regards the customer as an integral part of the project. Learning by the team and by the customer can be reflected in the software, because the specification of the problem is continuously refined during the development process.

Business changes: XP reduces the amount of change during the development of a single release due to the shortened release cycle. During a release, the customer is welcome to substitute new functionality for functionality not yet completed.

False feature rich: This risk is avoided since XP calls only for the implementation of the highest-priority tasks.

Staff turnover: This is often the biggest problem in software development projects, because programmers drop out due to job dissatisfaction and burnout syndromes. XP encourages human contact among the team thus reducing the loneliness that is often at the root of job dissatisfaction. It also asks programmers to accept responsibility for estimating and completing their own work, gives them feedback about the actual time taken so their estimates can improve, and respects those estimates. Clear rules for who can make and change estimates are established in order to avoid the frustration of a programmer when asked to do the obviously impossible. New team members are encouraged to gradually accept more responsibility, and are assisted along the way by experienced programmers.

XP allows programmers to work on tasks that really matter every day, since they are responsible for all aspects of the software. It avoids tempting programmers to make decisions they are not best qualified to make. On the other hand, it gives them the freedom to make decisions that they can make best. To customers and managers, XP promises to deliver the highest possible value out of every programming week. On top of that, they will be able to see concrete progress of the project every few weeks and they have the opportunity to influence the direction of the project in the middle of the development process without exorbitant cost increases.

The major disadvantage of XP is that its suitability is limited to relatively small software development projects. But if appropriate, XP promises to reduce project risk, improve responsiveness to business changes, improve productivity throughout the life of a system, and add fun to building software in teams.

2.3 Extreme Programming and the Flagpole project

The Flagpole project required software development for a variety of project areas like data collection, data processing, data compression, data publishing, and, of course, development of educational software, the area on which this thesis focuses. We did not really make use of a particular software development process model, due to a number of reasons. Educational software development in the Flagpole project was more about creating small software products that are relatively independent of one another than building a single large piece of software. Because of that, the impact of integration issues was lowered hence reducing project risks and constantly monitoring the software development process were not as relevant as they are in larger coherent software projects. Additionally, most of the people that developed the educational software for the Flagpole project were initially just programming 'greenhorns'. In the beginning, they did not have much experience with the different software development process models and all associated activities like requirements gathering, system design, testing, and implementation. Consequently, it would have been difficult to go with a sophisticated model that is aimed at experienced software developers. The main goal was to gain programming knowledge and develop relatively simple yet useful software for educational purposes.

However, extreme programming is as close to a software development process model used for the educational software in the Flagpole project as it gets. We applied numerous features that are characteristic of the XP paradigm as described earlier. First of all, we were able to do it because our task was not to build a large, complex system that requires a lot of manpower and resources, for which XP is not appropriate. In the beginning, our project team faced vague and eventually changing requirements, because none of us had ever built educational software before and we did not know how much time we would be able to spend on it on top of the course work that the

Master of Engineering Program required. We did not establish an explicit testing strategy before coding started, but all software was tested several times a day and integration was also frequently completed. Although we did not work as pairs of programmers sitting in front of and working on the same screen, there was constant code review by fellow teammates and collaboration took place during all phases of the software development process. This collaboration was supported by an open workspace in a computer laboratory at MIT where we did our project work for most of the time. The computers in this room are set up in an unobstructed way that facilitates communication between developers, a great help especially when the team structure consists mainly of inexperienced programmers. This setting helped to bring up fresh ideas for all software products that could be implemented and tested either immediately or at a suitable point of time. Since we did not really have customers that could give us feedback and help in refining the requirements, the teammates also filled up this role for one another. All of us started off with simple designs that constantly evolved over time as functionality was added and unneeded complexity was removed. We put minimal systems into production quickly and grew them in whatever direction seemed to be most valuable for a learning student, our assumed customer. No team member was forced to become a specialized analyst, architect, programmer, tester, or integrator – everyone participated in all of these crucial activities almost every day.

Software development in the Flagpole project was distinguished from software development done by companies in the information technology business by its research-based nature. A definite distinctive feature was that we did not have to take costs of software development into consideration, because we were all doing this work as our research project and we did not get paid for it. Deadlines were also not a big issue. Naturally, we had to be finished at some point of

time before the end of the project, but it cannot be compared to a strict deadline that eventually puts a lot of pressure on professional software developers.

3 Educational Software in the Flagpole project

3.1 Overview

As mentioned earlier, the development of educational software was one of the main aspects of the Flagpole project. The goal was to improve several facets of structural engineering education, from the learning student's understanding of structural engineering concepts over his ability to use software tools when solving engineering problems to the teacher's understanding of how students learn most efficiently. I want to especially thank all the group members of the Flagpole project team who developed these applets and allowed me to use them for this document.

The Flagpole project team decided to use Java applets for the development of the educational software. There are many reasons for this decision. An applet is a program written in the Java programming language that can be embedded in an HTML page and thus it can be displayed on the Internet. It is not necessary for the customer to obtain expensive additional software to be able to use these applets, because all he needs is a Java plug-in that can be downloaded free of charge. A page that contains an applet can be viewed using a Java technology-enabled browser. The applet's code is transferred to the user's system and executed by the browser's Java Virtual Machine (JVM). This was very important for us because our philosophy is to make our software available to anyone who is interested in it, anywhere in the world. Java is a widely used and powerful object-oriented programming language. It provides a great diversity of graphical tools that have been proven very helpful in the visualization of engineering concepts. In general, computer users like the simple, click-to-start model of Java applets. IT managers like the flexibility of simple network distribution. And software developers are pleased that Java

technology speeds up their work and enables solutions that could not be created any other way. Usually, Java applets run with high reliability and consistency. However, they are subject to security restrictions of browsers that try to keep them from compromising system security. Any applet that is loaded over a network cannot load libraries or define native methods, ordinarily read or write files on the host that is executing it, make network connections except to the host that it came from, start any program on the executing host, or read certain system properties. On the other hand, applets do have some capabilities that applications do not have. They can invoke public methods of other applets on the same page, make network connections to the host they came from, and, when running within a web browser, cause HTML documents to be displayed. Moreover, applets that are loaded from the local file system of a computer have none of the restrictions that applets loaded over the network have.

The educational applets developed for the Flagpole project provide a good opportunity for students to deepen their knowledge of structural engineering principles gained in lectures or start learning these principles on their own. They cover several areas of structural engineering, but the limited scope of the project and the restricted time resources certainly did not allow us to cover the complete range of this science. All applets developed during the project offer the user the opportunity to gain hands-on experience about possible abstractions of real-world systems and about the influence of certain parameters on the results of a structural analysis. The underlying philosophy is that the student has as much influence on the setup of the structural system as possible in order to be able to explore the effects of changing parameters on system behavior. Additionally, on-line help and tutorial sections are provided, which contain the theoretical background of the concepts. Little initial knowledge is required before starting to use the software, although it is probably not suitable for beginners.

This chapter explains the applets about Mohr's circle (tutorial and real-time), a single degree of freedom system, and a tuned mass damper. The focus is a little bit more on the resulting software than on the mechanical concept that is portrayed. Yet some information is provided about the foundational engineering concepts, the implementation of those concepts into software, and screenshots of the applets that show the working products.

One of the Java applets that have been developed for the Flagpole project, the one about the Culmann Method, is not part of this chapter, because it is explained in more detail in the next chapter to illustrate all aspects of the development of educational software in greater depth.

3.2 Mohr's Circle

Mohr's circle is a popular graphical tool that helps to determine the normal stress and shear stress acting on a plane. Introduced by the German engineer Otto Mohr in 1882, it was the leading method to visualize maximum stresses before hand-held calculators became available.

The abscissa of the graph where the Mohr's circle is drawn represents the normal stress, and the ordinate represents the shear stress. The circle is centered at the average stress value and its radius is equal to the maximum shear stress. The normal stresses are called principal stresses when the stress element is aligned with the principal directions, i.e. the horizontal axis of the Mohr's circle's coordinate system. The principal stresses and the maximum shear stress are obtained immediately after drawing the Mohr's circle, the greatest advantage of this graphical tool. It can also be used to transform stresses from any desired coordinate set to another, or to illustrate strains in a plane instead of stresses. To draw the Mohr's circle to represent the two-dimensional

stress state at a point, it is sufficient to know the normal and shear stresses acting on two perpendicular planes at that point. Initially, these given values of stress are plotted as two pairs of coordinates of (normal stress, shear stress). A line is drawn to connect the two points in the following step. The midpoint of this line is located on the x-axis and it is the center of the corresponding Mohr's circle. Finally, the Mohr's circle is drawn, and principal stresses can be taken from the circle as well as maximum shear stresses. The applet about Mohr's circle follows these steps, except that it uses transformation equations to calculate the characteristic points of the circle from the given values. The following screenshot shows the output of that applet:

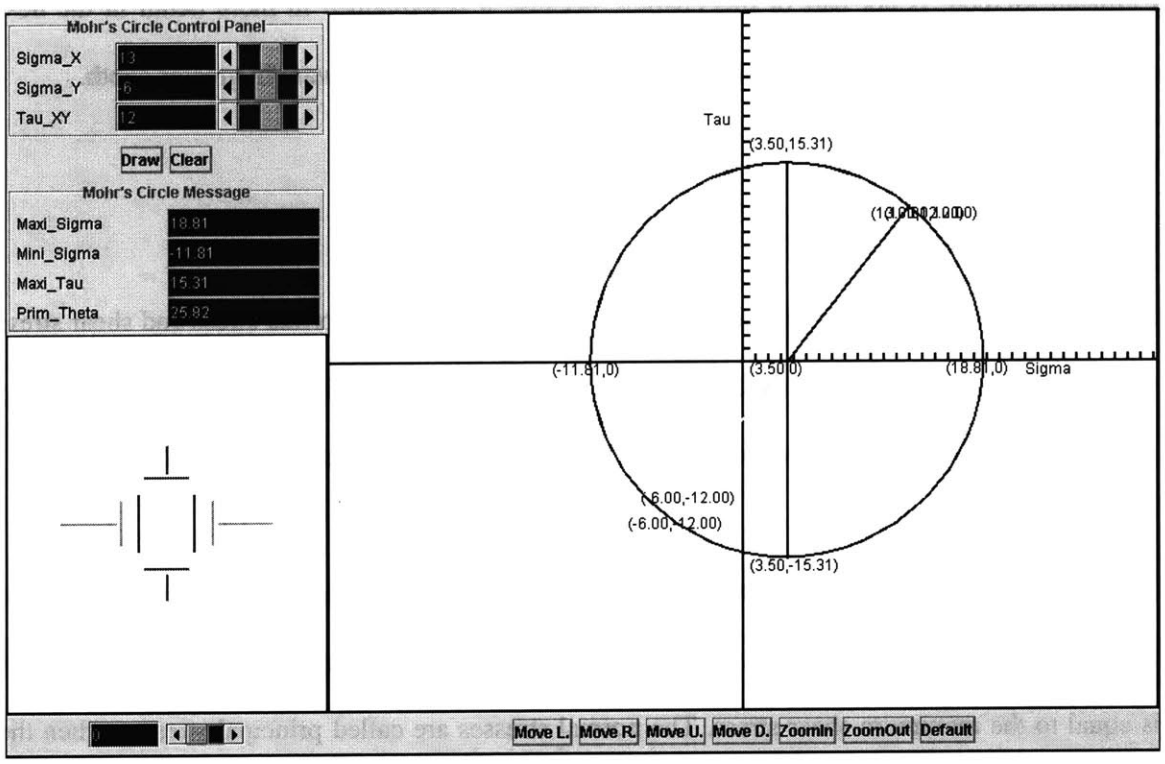

Figure 3-1: The Mohr's Circle applet and its results

When the user starts the program, he enters values for the normal stresses σ_x and σ_y as well as for the shear stress τ_{xy}, all of them referring to the initial coordinate system of x and y. When he prompts the applet to draw the circle, the circle is displayed. The principal stresses σ_1 and σ_2, the

maximum shear stress τ_{max} and the angle θ_p, which indicates the rotational difference between the initial axes and the principal axes, are calculated and displayed on the screen. The user can determine the stress values for any rotated coordinate system by using the tool on the lower left-hand corner of the screenshot to dynamically change the rotation angle. The angle and the corresponding stress values are immediately shown as part of the Mohr's circle and the changing size of these stresses can be observed in the lower left-hand part of the screen. The applet allows the user to select the drawing properties he is interested in. He can choose to display the pole position, multiple circles for multiple points or stress states in one coordinate system, or he can decide whether he wants to see the characteristic numbers of the Mohr's circle on the screen or not. In case the resulting Mohr's circle is too small, too big, or too far off on one side of the screen, the applet provides options to zoom in, zoom out, and move the Mohr's circle to any desired direction.

People who are not familiar with the concept of Mohr's circle may be able to use this program by punching in random numbers, but they will not be capable of making sense of the results. Therefore, this applet offers a help section that gives background information about the origin and intention of Mohr's circle and explains its methodology. But since the educational software in the Flagpole project has been developed with focus on university students, a basic understanding of engineering concepts has been assumed. A beginner may not be able to use this program as a starting point, but, going back to our intentions mentioned earlier in this chapter, our goal was to create tools that support learning from lectures and do not replace it. The following screenshots offers an insight into the applet's help section:

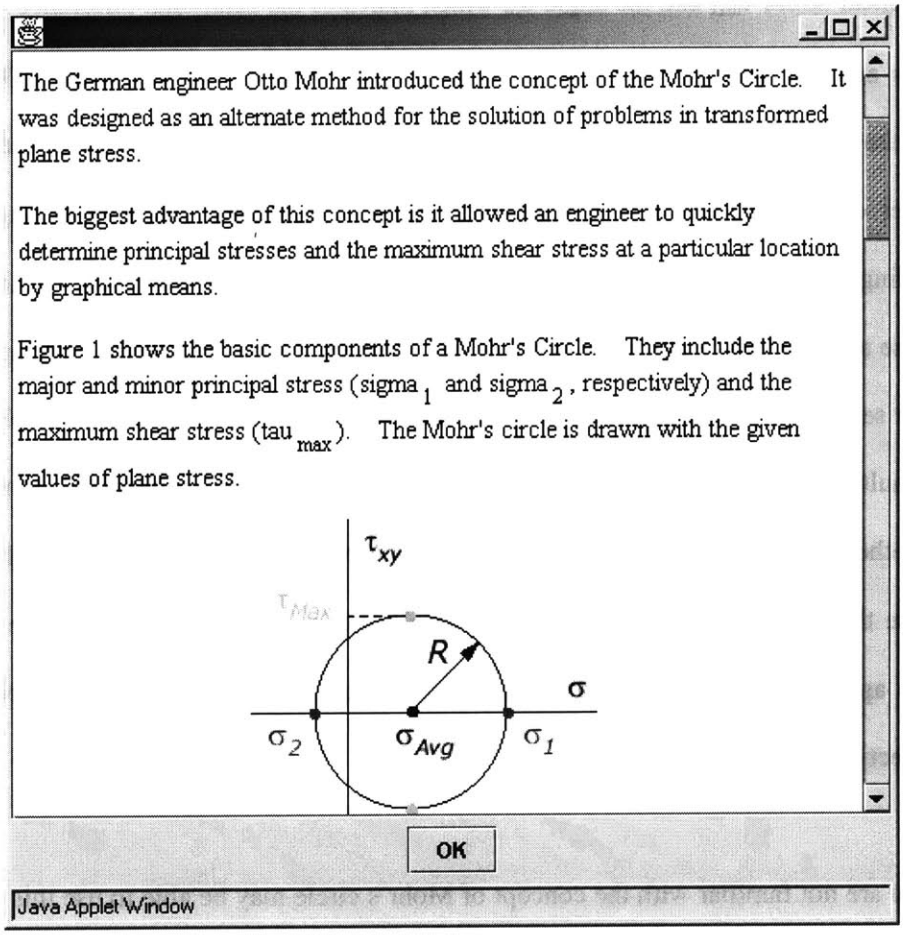

Figure 3-2: The help section of the Mohr's Circle applet

Although one might argue that learning a tool like Mohr's circle is done most effectively by drawing every step by hand, thus thoughtfully following the process, this applet provides the opportunity to change parameters interactively and study several stress states, an activity that would otherwise take a lot of time or it would not be done at all.

A drawback of the Mohr's circle approach is that the exactness of the results is lower when drawn by hand than when calculating the stresses. Nevertheless, Mohr's circle is a valuable and vivid engineering tool. It is worth mentioning that this applet is already used as a teaching tool in an undergraduate course at MIT.

3.3 A Single Degree of Freedom System

A single degree of freedom system (SDOF) with free vibrations (no external forces) is the most basic system for a dynamic structural analysis. It restricts possible movements to just one direction and abstracts the actually evenly distributed mass of the structure to a single lumped mass. Naturally, these simplifications limit the applicability of this analytical method to relatively unsophisticated systems. But that is not a big problem since our abstractions are on the safe side and the great reduction of work absolutely justifies these simplifications. Concluding, we can say that for a restrained cantilever, which is the common system used in the Flagpole project, this is a valid method.

For an unforced damped SDOF system, the general equation of motion, which describes the system state, is given by,

$$m\ddot{x} + c_v\dot{x} + kx = 0$$

where m stands for the mass, $\ddot{x}$ for the acceleration of the mass, c_v is the viscous damping variable, $\dot{x}$ is the velocity of the mass, k represents a spring stiffness, and x indicates the displacement of the mass. The goal is to solve this equation in order to forecast the displacement of the mass at any point in time, if the initial conditions and the system properties are known. The initial displacement is defined as $x(t=0) = x_0$ and the initial velocity is described as $\dot{x}(t=0) = v_0$. This equation of motion is a second order, homogeneous, ordinary differential equation (ODE). We assume that all parameters (mass, spring, stiffness, and viscous damping) are constant, which turns the equation of motion into a linear ODE with constant coefficients. It can then be solved by the Characteristic Equation method. The characteristic equation is,

$$ms^2 + c_v s + k = 0$$

which determines the two independent roots for the damped vibration problem. The roots to the characteristic equation fall into one of the three following cases:

1. If $c_v^2 - 4mk < 0$, the system is termed *underdamped.* The roots of the characteristic equation are complex conjugates, corresponding to oscillatory motion with an exponential decay in amplitude.
2. If $c_v^2 - 4mk = 0$, the system is termed *critically-damped.* The roots of the characteristic equation are repeated, corresponding to simple decaying motion with at most one overshoot of the system's resting position.
3. If $c_v^2 - 4mk > 0$, the system is termed *overdamped.* The roots of the characteristic equation are purely real and distinct, corresponding to simple exponentially decaying motion.

The Java applet about the single degree of freedom system only deals with underdamped systems, because this is the most common and meaningful case.

To simplify the solutions coming up, we define the critical damping c_c, the damping ratio ζ, and the damped vibration frequency ω_d as,

$$c_c = 2m\sqrt{\frac{k}{m}} = 2m\omega_n$$

$$\zeta = \frac{c_v}{c_c}$$

$$\omega_d = \sqrt{1-\zeta^2}\,\omega_n$$

where the natural frequency of the system ω_n is given by,

$$\omega_n = \sqrt{\frac{k}{m}}$$

$\zeta < 1$ or $c_v < c_c$ are representative of an underdamped system, as mentioned earlier the only case implemented in our software. The resulting displacement solution for this kind of system is,

$$x(t) = e^{-\zeta\omega_n t}\left[x_0 \cos(\omega_d t) + \frac{v_0 + \zeta\omega_n x_0}{\omega_d} \sin(\omega_d t) \right]$$

This is the algorithm that has been used in the applet's code to produce the displayed results. Having described the theoretical background of a single degree of freedom system, the following paragraphs will discuss its implementation into a piece of software, its difficulties, peculiarities, and its usefulness.

After uploading it from the Flagpole website, the Java applet about the single degree of freedom system allows the user to choose one of the most frequently used cross sections: a rectangle, a circle, or an I-section. Regardless of the cross section type, the user has to enter the length, the weight, and the damping ratio of the structure. He has to be careful when entering the value for the damping ratio, because it is entered as a percentage, which means that the damping ratio $\zeta = \frac{c_v}{c_c}$ has to be multiplied by one hundred in order to get meaningful results from the program. The reason for that lies in the common habit of specifying the damping ratio as a percentage. Additionally, the user can provide values for the initial conditions, displacement and velocity at time zero at the top of the system. Depending on the type of cross section that is selected, the user can input the additional cross section's properties. For a rectangle, width and height are required. A circular cross section asks only for a radius. The width and thickness of the flange and the height and thickness of the web characterize the I-section. If both initial displacement and

velocity are set to zero, nothing will happen, because that is no longer a dynamic problem. There is no option to enter a force directly on the structure, but it is possible to transform a force at any place of the structure to a displacement at the top with a basic structural calculation that has to be conducted by hand. Usually, the weight of the structure has to be calculated by hand, too, due to the fact that in most cases the length of the structure and the density of the material are given. However, it is important to remember that this applet does not handle the vibration of a single degree of freedom system under a general forcing function $f(t)$, which makes solving the equation of motion more complex and requires the use of either a convolution integral or a Laplace transform.

The following screenshot shows the interface of the applet with the produced output after the parameters had been set. In this case, an I-section had been chosen.

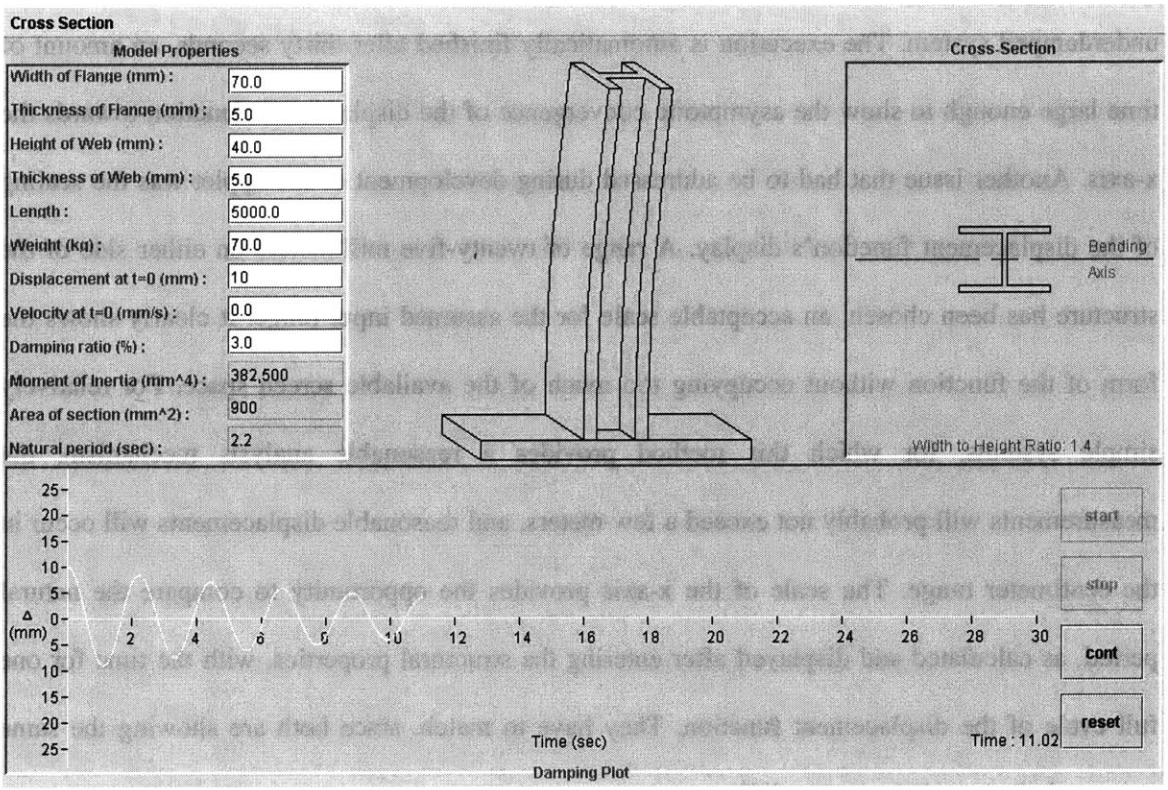

Figure 3-3: The Single Degree of Freedom applet

The applet keeps all inputs and results conveniently visible all the time. As you can see, the execution can be stopped, continued, and reset at any point of time. The applet calculates and displays the moment of inertia, the area of the cross section, and the natural period of the structure from the provided input. In addition, the width-to-height ratio is computed, but only for a rectangle and an I-section. When the user hits the start button, the applet starts to show the moving cantilever, and, simultaneously, the development of the displacement function, labeled as Δ, in the lower part of the screen. Underlying is the solution of the equation of motion, executed by the program according to the procedure described above. The spring stiffness is calculated depending on the system's length, the cross section's moment of inertia, and the material's modulus of elasticity, for which the value is predefined in the applet's Java code. The display fittingly shows the exponential decay of the displacement amplitude that is characteristic for an

underdamped system. The execution is automatically finished after thirty seconds, an amount of time large enough to show the asymptotic convergence of the displacement function towards the x-axis. Another issue that had to be addressed during development of the applet was the scaling of the displacement function's display. A range of twenty-five millimeters on either side of the structure has been chosen, an acceptable scale for the assumed input range. It clearly shows the form of the function without occupying too much of the available screen space. For relatively simple systems, for which this method provides a reasonable analysis mechanism, the measurements will probably not exceed a few meters, and reasonable displacements will occur in the centimeter range. The scale of the x-axis provides the opportunity to compare the natural period, as calculated and displayed after entering the structural properties, with the time for one full cycle of the displacement function. They have to match, since both are showing the same structural characteristic, just in different ways.

The Java applet for a single degree of freedom system does not take into consideration material constraints like the ultimate strength that might lead to system failure, thus naturally ending the dynamic process. Accordingly, the applet does not have an option for the user to select a desired material with specific material properties. This means that the user cannot make any changes to the modulus of elasticity directly. If he wants to change the system behavior that corresponds to a change of the modulus of elasticity, he can only take that into account indirectly by adjusting the cross section measurements. This will change the moment of inertia, which is multiplied with the modulus of elasticity in the formulas leading to the solving of the equation of motion. Naturally, these restrictions limit the applet's applicability to systems and circumstances where system failure cannot occur.

A student who uses this applet to learn more about single degree of freedom systems or structural vibrations can run many different systems through the program in a short period of time. Carefully studying the changing results in relation to changing parameters will greatly improve his mechanical understanding. The goal of this program is to enable the student to get a feeling for the effects of varying dimensions or of choosing a certain cross section. Finally, he should be prepared to roughly predict the impacts of parameter changes on the systems behavior and to choose appropriate cross section types and dimensions for real-world structures.

Of course, simply playing around with this applet will not achieve these goals. Like any computational implementation of an engineering concept, this applet has its limits that the user has to be aware of, as mentioned above. It is subject to the same philosophy and the resulting assumptions as all the Flagpole software. Intended to be a teaching support tool for students who gain the knowledge of the underlying theory through lectures and assignments, this applet is probably not helpful for the beginner. Nevertheless, even with just a small engineering background and an arising grasp of dynamic concepts, students are able to enhance their understanding significantly.

3.4 Simulation of a Tuned Mass Damper

The Java applet about the tuned mass damper, also known as a vibration absorber, takes the basic concept of a single degree of freedom to the next level, a more complex two degree of freedom system. A tuned mass damper (TMD) is a mechanical device used to decrease or eliminate unwanted vibration. It is a passive damping device. A TMD consists of an additional mass attached to the main structure by means of an elastic spring and a damping element. Its most frequent application is to synchronous machinery, operating nearly at constant frequency, for the

vibration absorber is tuned to one particular frequency and is only effective over a narrow band of frequencies. However, absorbers are also used in situations where the excitation is not a harmonic one. One common use of tuned mass dampers is the reduction of wind-induced vibration of tall buildings when the motions have reached an annoying level for the occupants. The following paragraph deals with the basic ideas behind a tuned mass damper.

The equations of motion for the main mass and the absorber mass are the same as for a two degree of freedom system. Since the system has two degrees of freedom, two resonant frequencies exist, and the response is unbounded at those frequencies. The natural frequency of the mass damper is tuned to the natural frequency of the main structure. The usefulness of a tuned mass damper becomes obvious if one compares the frequency-response function of a system containing a vibration absorber with the response of the main mass alone. If the system-exciting frequency is equal to the natural frequency of the main mass, the response amplitude of the main mass alone is unbounded but is zero with the presence of the absorber mass. Thus if the exciting frequency is close to the natural frequency of the main system, and operating restrictions prohibit the variation of either one, the main system's response amplitude can be reduced to near zero with the help of a vibration absorber. For this purpose, the absorber system should exert a force equal and opposite to the exciting force. Choosing the right absorber mass is potentially problematic. A large absorber mass presents a practical implementation problem, but at the same time, a smaller mass will narrow the operating frequency range of the tuned mass damper.

The Java applet about the tuned mass damper developed for the Flagpole project demonstrates the dynamics of this vibration absorber. The main building is modeled as a lumped mass on the

highest floor and the damper is modeled as a mass on rollers, which is attached to the main mass by a spring. The figure below shows the executing applet.

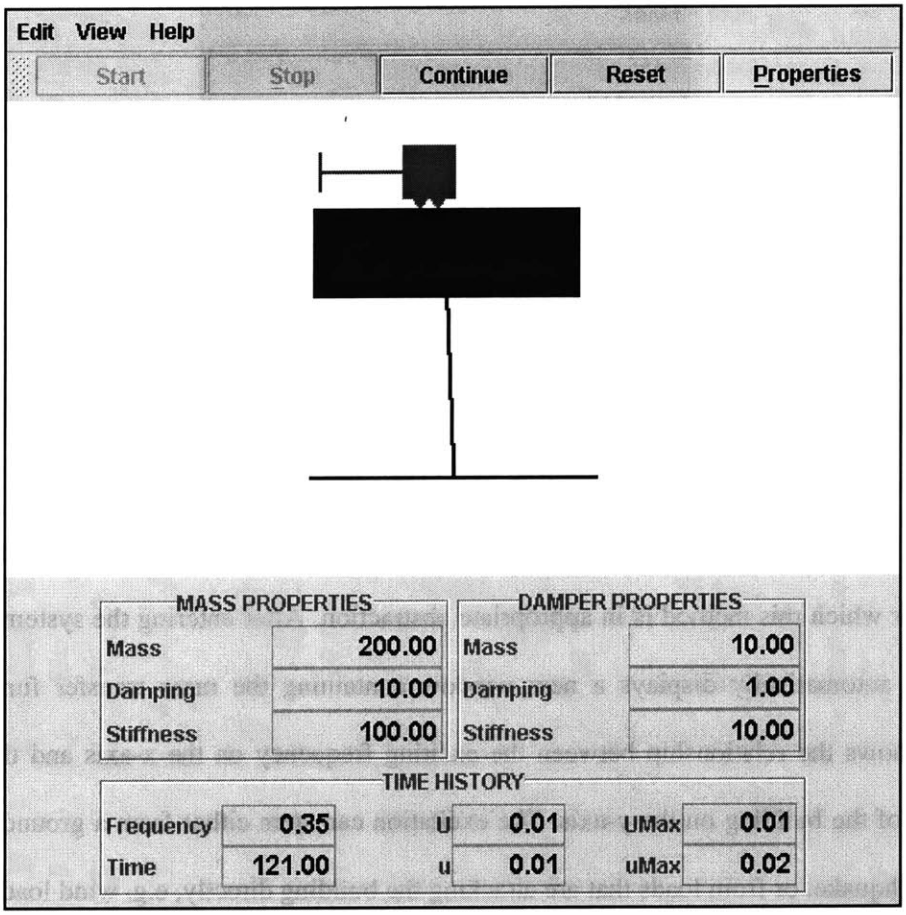

Figure 3-4: The Tuned Mass Damper applet

The applet allows the user to change all parameters of the system. In a separate window, triggered by a click on the 'System properties' menu item under the 'Edit' menu, the user inputs the mass, the stiffness, and the damping of both main mass and damper.

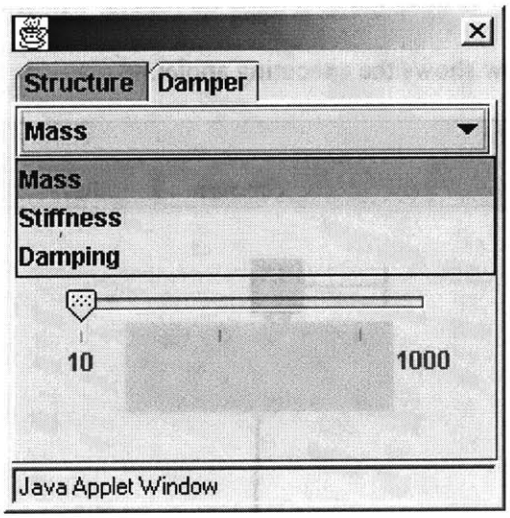

Figure 3-5: System Properties window

The range of the system properties is limited to a meaningful scale, i.e. it allows simulation of systems for which this method is in appropriate abstraction. After entering the system properties, the applet automatically displays a new window containing the mass transfer function. This function shows the relationship between the exciting frequency on the x-axis and the response amplitude of the building on the y-axis. The excitation can stem either from a ground excitation, e.g. an earthquake, or from loads that are attacking the building directly, e.g. wind loads.

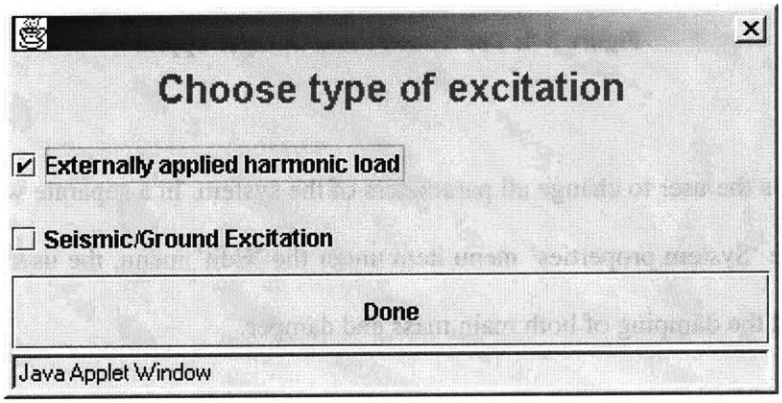

Figure 3-6: Types of Excitation

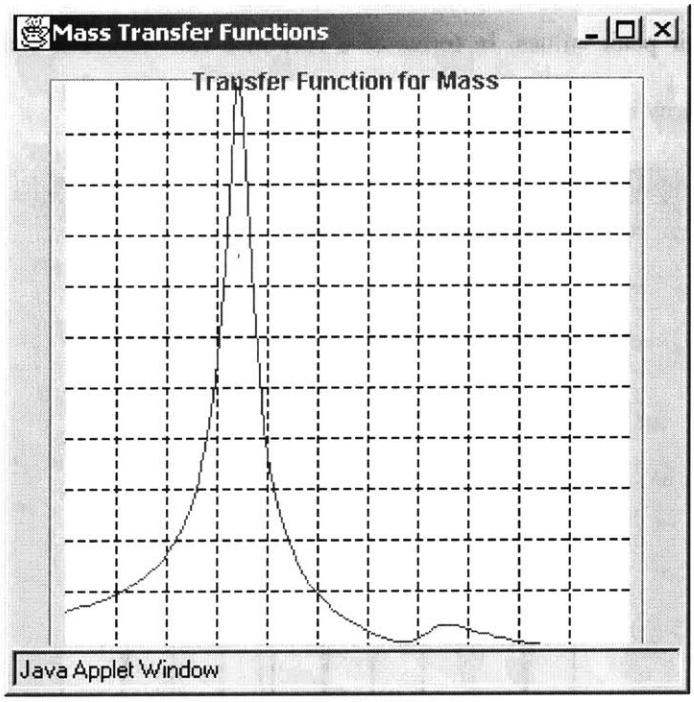

Figure 3-7: Mass Transfer Function

This window allows the user to change the excitation frequency during run time of the applet. Bringing it to the forefront and moving the mouse pointer over the curve lets the user choose the excitation frequency for the system with a mouse click. This is an excellent opportunity to study the system response under varying circumstances.

The user can now start the simulation of the system. This applet is different from the single degree of freedom applet in not restricting the execution time. In addition to the running execution time, it displays the system properties, the excitation frequency, the current displacement of the main mass (U) and the damping mass (u), and the maximum displacements of both masses on the main screen. Furthermore, a new window pops up that shows the steady state response for the main mass. In the steady state the system oscillates, i.e. the position, velocity, and acceleration exhibit oscillatory behavior. It is useful to look at the amplitude of these

oscillations, i.e. their peak values, in terms of a response function. The response function of a system determines how it responds to various driving frequencies.

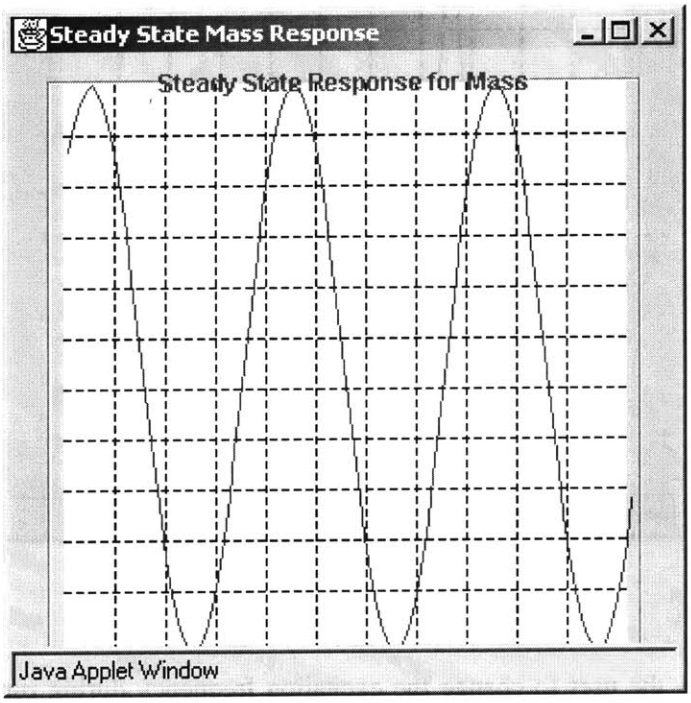

Figure 3-8: Steady State Response window

In case the user wants to learn more about the concept of a tuned mass damper or wants to know how to operate the applet, a help section is provided, which also contains some questions to ponder about while trying to understand the basic idea of a tuned mass damper.

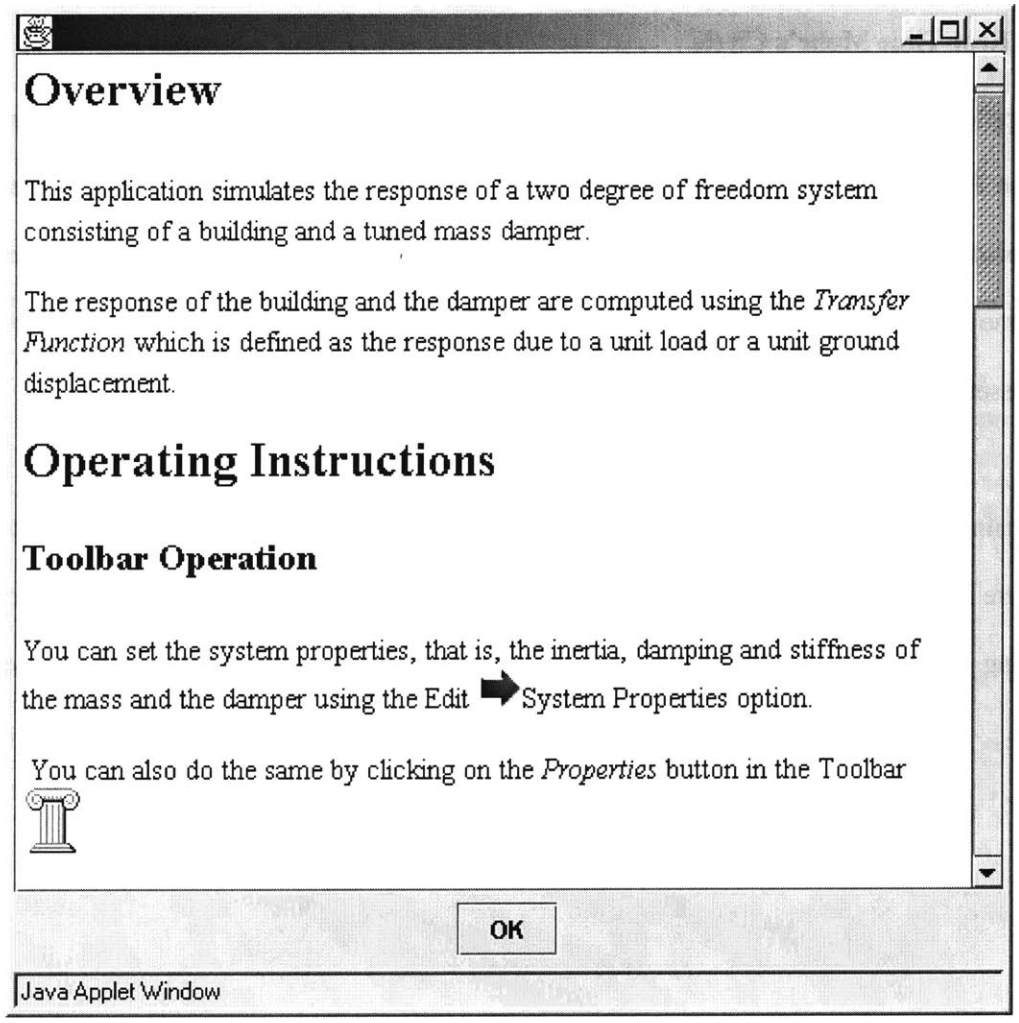

Figure 3-9: Help section of the Tuned Mass Damper applet

This applet gives students a very useful tool to study the effects of changing system properties in an application that is very close to actual building problems. Naturally, this applet contains some simplifications, which might make it inappropriate for real-world structures, but it can be a good starting point for a rough analysis to approximately estimate system behavior.

3.5 Real-Time Mohr's Circle

This applet builds on the introductory applet about Mohr's circle that was explained earlier in this chapter. For this reason, the underlying mechanical concept of Mohr's circle is not repeated here. The focus of the following paragraphs is on the difference between an introductory applet and a real-time applet about the same topic. Since both of these applets portray the same basic concept, their essential structure is very similar.

The main addition in the real-time applet is the opportunity to display the actual stresses of a structure in real-time. It still contains almost all the features of the introductory applet, except the zooming capabilities. The following screenshot shows the similarity between both software pieces.

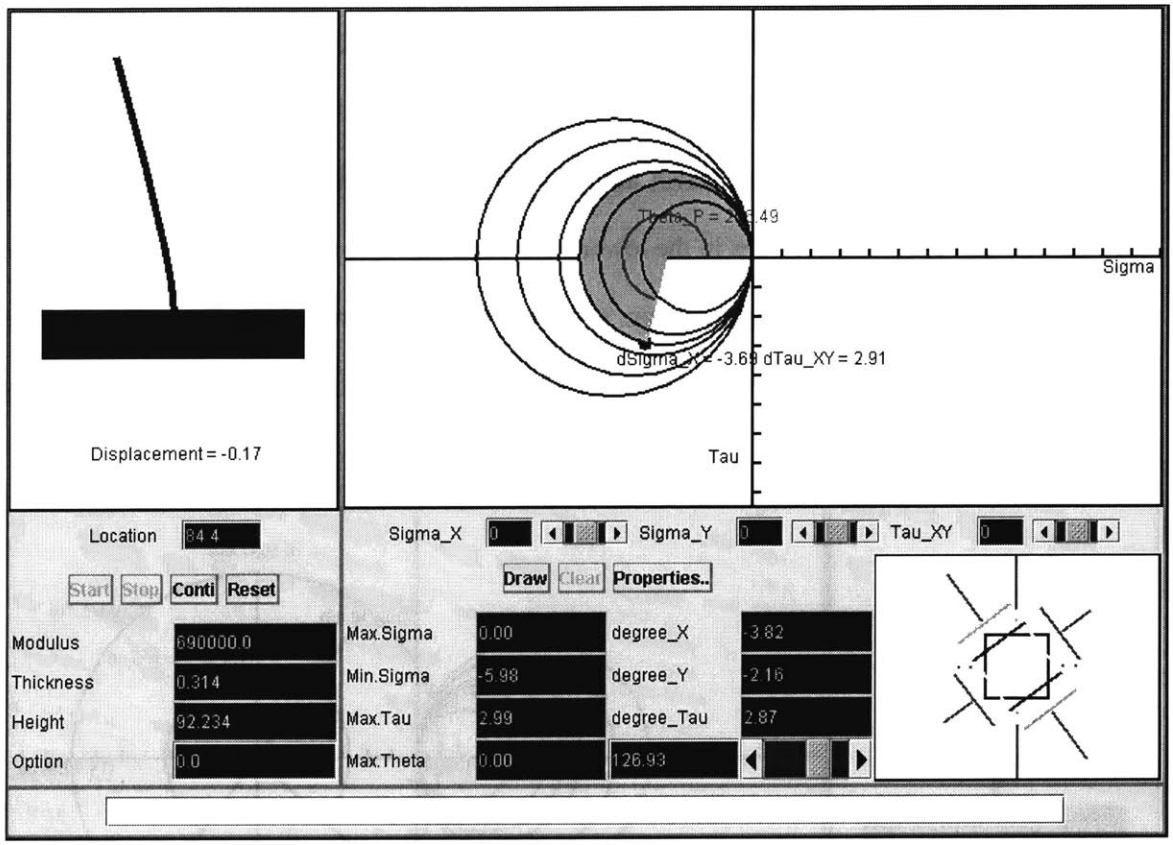

Figure 3-10: Real-Time Mohr's Circle applet

It is possible to drag on the Mohr's circle to see the stress change of an element corresponding to the degree of rotation.

The screen is divided into four main parts. The part at the upper left hand side of the screen shows an upright cantilever, the mechanical abstraction of the flagpole and the model we have been using for testing purposes. Clicking on any point of the cantilever will determine the point of the structure for which the Mohr's circle is going to be displayed. The left part of the screen is also used for the output of the displacement at the top (the basis for calculating the stresses), the location on the cantilever that corresponds to the Mohr's circle, and the actual system properties on the lower part. It is possible to stop, continue, and reset the applet at any point of time, since

execution will not stop automatically. The upper right hand part of the screen is reserved as a drawing area for the displaying of the circles after starting the applet. In addition to viewing a number of Mohr's circles that are drawn automatically corresponding to the incoming sensor data, the user can enter stress values in the lower right hand part to compare the actual stress state with any desired situation. The figure below shows this kind of comparison. It also enables the user to see the applet's option to rotate the coordinate system in order to examine interesting areas.

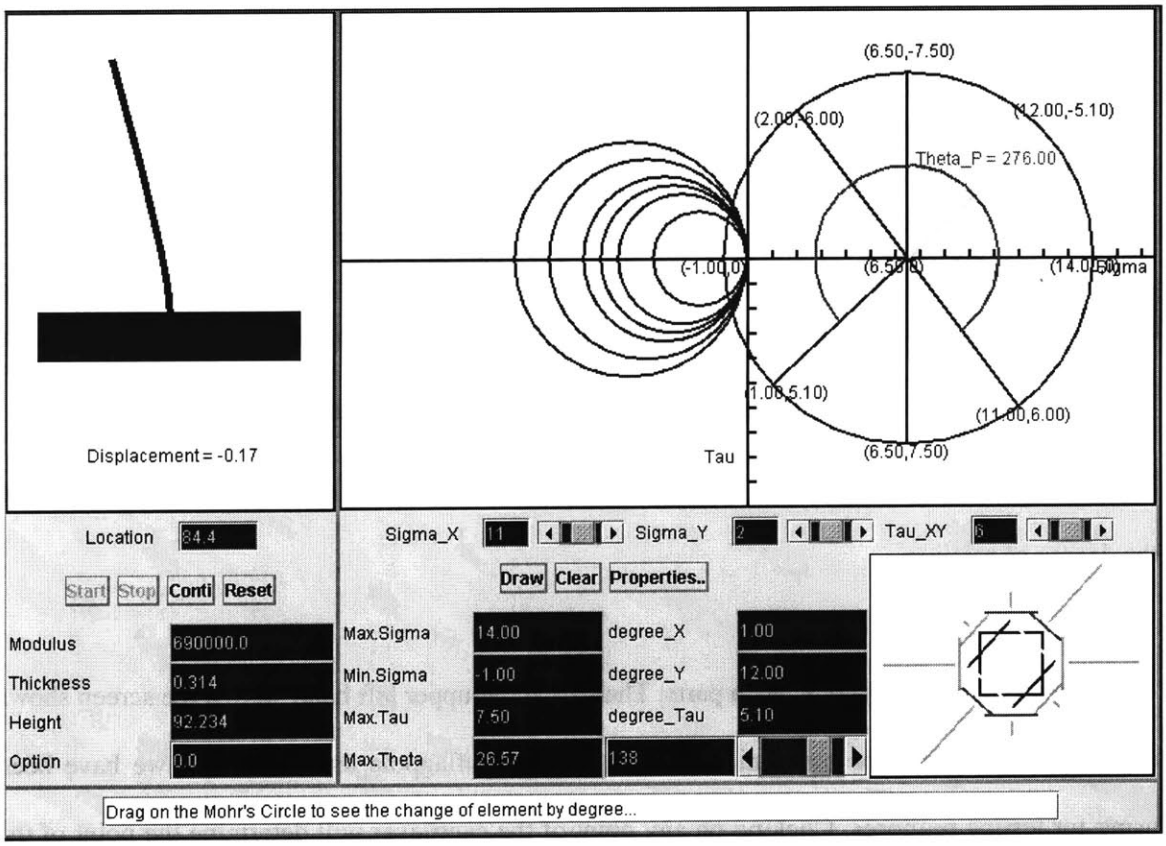

Figure 3-11: Comparison of real-time and fictitious data

This applet links the theoretical background of mechanical principles to real-world systems and helps the students get an idea of how to apply the knowledge gained from lectures and readings in a specific situation.

At the current stage of the project, all real-time information is extracted from the sensors on the aluminum model. But once the flagpole itself is equipped with the sensing devices tested at the model, these applets can also be used to monitor the flagpole's behavior without major complications.

4 An Example: The Culmann Method

The previous chapters introduced the Flagpole project, explained different software development process models, and gave an overview of the educational software developed for the Flagpole project. In this chapter, I am elaborating on an educational Java applet about a graphical tool for mechanical principles in more detail that was also created during that project. Starting with an explanation of the Culmann method's theoretical background, the implementation of this method in a Java applet is illuminated as well as the resulting software product. The complete code can be found in the appendix section of this document.

4.1 Explanation of the Culmann Method

The Culmann method is a graphical tool to solve reaction forces of statically determined systems. Its basic suitability is limited to systems with three reaction forces, and no restrained systems with reaction moments can be solved. There are ways to apply this tool to more complex systems, but they are not implemented in this software that shows the basic concept of the Culmann method. Although it is a relatively old tool, it is still very useful to visualize the solving of reaction force problems. The theoretical background of this method is explained in the following section.

As mentioned above, the mechanical problem that the Culmann method is able to solve is that of a statically determinate system with three reaction forces. There are some rules that have to be followed for the setup of such a system to be statically determinate. Essentially, the three reaction forces are not allowed to be parallel and their lines of application must not intersect at only one

point. No further rules have to be followed, i.e. it is possible for the system to contain a bend or to be curved.

The methodology of the Culmann method considers only one outer force. Nevertheless, it is also feasible to solve statically determinate systems with more than one outer force. In this case, preliminary work has to be done before applying the Culmann method. All the outer forces working on the system have to be united to a resulting single outer force with the same size and direction as the combination of all the outer forces. There are several methods to achieve this goal, graphical methods as well as computational methods. A popular graphical method is the 'rope hitting a corner procedure'. These methods are not explained here, they can be found in appropriate literature.

The following figure shows one example of a statically determinate system for which the Culmann method is applicable. It is a beam with two supports, one of them is a fixed bearing and the other one is a floating bearing. There are no outer forces shown in this figure. As long as the above-mentioned conditions for the reaction forces are met, the supports can be located anywhere on the system.

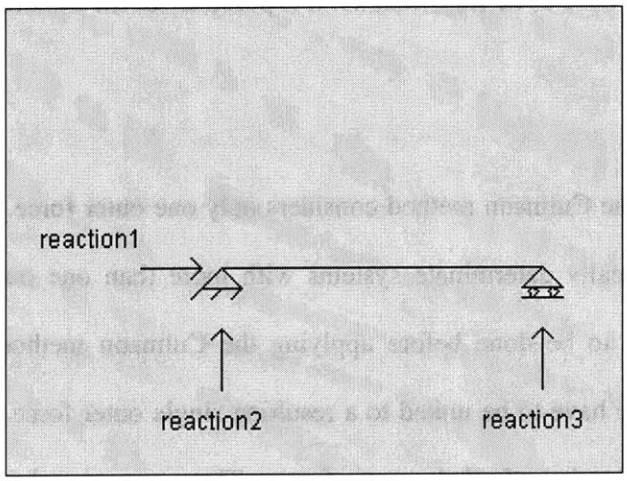

Figure 4-1: Example of a statically determinate system

Many other systems for which the Culmann method is suitable can be envisaged, but this system is probably the most common and the most straightforward one to learn this tool. Therefore, it is also the system that I chose to implement in the Java applet.

If an outer force is applied to the system, it has to be taken up by the three reaction forces in order to maintain equilibrium of the forces. Since we are not considering dynamic problems here, maintaining a static equilibrium is the basic goal of all operations.

The Culmann method is an alternative to calculating how the three reaction forces take up the outer force, i.e. how big the reaction forces are.

The drawing of the system is called the 'layout plan'. It contains the system, the supports and the outer force. The figure below shows such a layout plan.

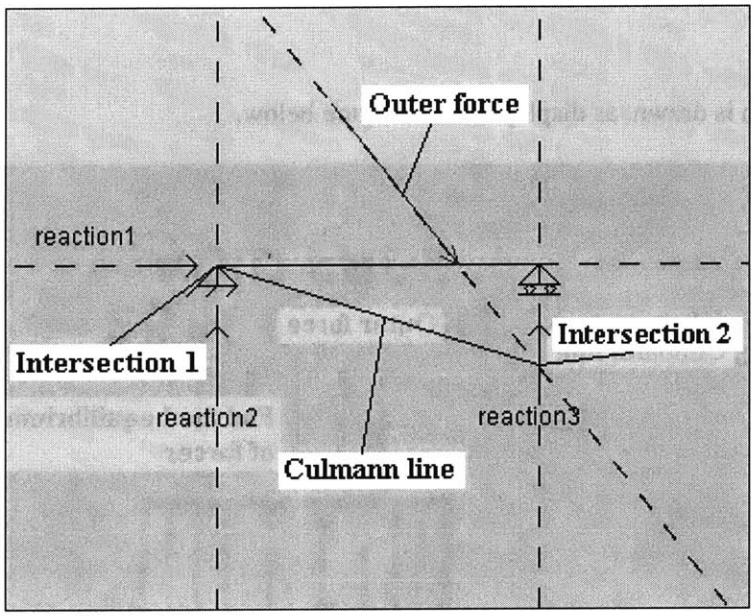

Figure 4-2: The Layout Plan of the Culmann Method

It is not necessary for the forces to be scaled in the layout plan, only their directions and positions are important. Besides, the size of each reaction force is not known at this stage. Each of these four forces is extended at both ends to display the line of application. A force's line of application is characterized by having the direction of the force and by the fact that it is possible to put the force on any point of this line without changing the equilibrium of forces in the system. It is now possible to determine two points of intersection (intersection 1 and intersection 2), each of them marking the intersection of two lines of application. Generally, it is one's own choice which lines of application to intersect, but it is not possible to intersect two lines if the remaining lines are parallel and do not have a point of intersection, because the Culmann method cannot work in that case. For instance, in the figure above it would not be allowed to intersect the outer force and reaction 1, since reaction 2 and reaction 3 are parallel and do not intersect. Connecting intersection 1 and intersection 2 will create a line that gave this method its name, the 'Culmann line'. At this point, the work in the layout plan is finished.

Now, a force plan is drawn, as displayed in the figure below.

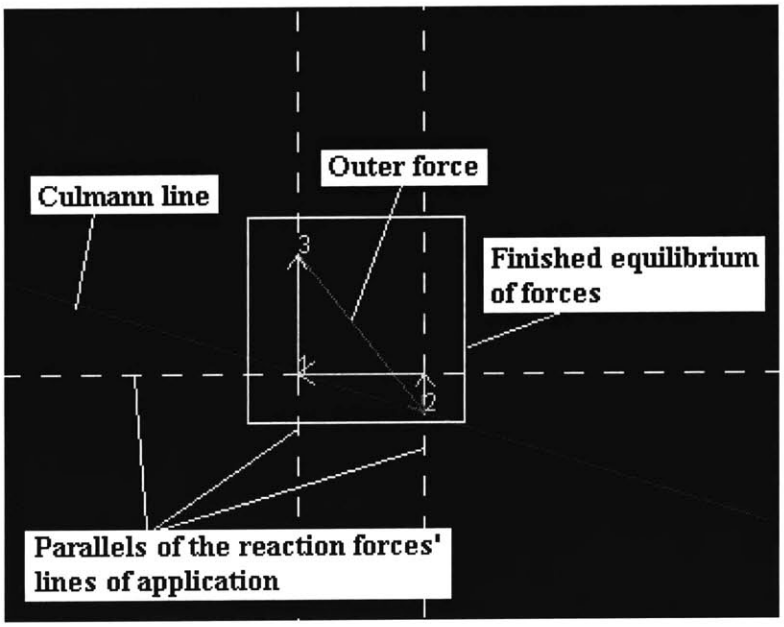

Figure 4-3: The Force Plan of the Culmann Method

It starts with drawing a line that is a parallel to the Culmann line of the layout plan. A force scale is set for the force plan, according to which the outer force of the system is drawn with its end anywhere on the Culmann line. A parallel of the line of application that was intersected with the outer force in the layout plan, i.e. reaction 3 in this case, is attached to the free end of the outer force in the force plan, i.e. the end that is not located on the Culmann line. The length of the parallel between the end of the outer force and the point where this parallel hits the parallel of the Culmann line determines the size of this force. At this stage, we have two intersection points on the parallel of the Culmann line in the force plan, and we still have to determine the size of the remaining two reaction forces that were intersected with each other in the layout plan. This can be achieved by attaching a parallel of each remaining reaction force to one of the two intersection points on the Culmann line. It does not matter which force is attached at which end, the results

will be the same, and only the drawing will look slightly different. Finding the intersection of these two parallels will close the parallelogram of forces and deliver the results for the last two reaction forces, reaction 1 and reaction 2. Now that a graphical equilibrium of forces has been created, the size of each reaction force can be measured in the force plan according to the scale factor and the problem is solved. One interesting thing to notice is that no matter what direction of a reaction force was chosen in the layout plan, the correct direction in which the reaction force works is automatically delivered in the force plan. In this example, that is what happened with reaction 1. It is drawn with direction to the right in the layout plan, but its actual direction is to the left because it has to take up the horizontal part of the outer force, as can be seen in the force plan.

4.2 Implementation of the Culmann Method in Java code

Implementing a graphical mechanics tool in executable code poses some special issues that have to be addressed. For instance, the Java programming language currently does not provide an opportunity to simply draw a line that is parallel to another line. Therefore, these issues have to be solved in a more indirect way. The basic structure of the Java applet about the Culmann method and its design issues are explained in this section. Since we are dealing with a Java applet that is going to be displayed on the Internet, a separate HTML file is needed that refers to the files containing the Java code.

The basic idea of designing the Java applet has been to simultaneously show as much information on the screen as possible without cluttering it up too much. The layout and force plans were to be displayed on the screen all the time in order to show all essentials of the Culmann method during the whole execution time. In addition, I decided to also display the characteristic data of the outer

force, i.e. its position, its size, and its direction. The user should be allowed to enter only one outer force. The applet should not be suitable for multiple outer forces since such cases are actually not suitable for the Culmann method, as mentioned earlier. Finally, I wanted a description of the series of steps that the Culmann method requires to appear on the screen simultaneously, because the focus of this software is on education.

Having made these decisions, they had to be transferred into Java code. I decided to use panels to hold each of the four major areas of the project as described above. The main panel holds and arranges all these four panels, which are programmed in separate class files. An animated introduction panel starts the program, from where the user can proceed to the main screen where all the actions concerning the Culmann method are performed. The figure below shows the classes of the applet and their structure. Connections between the classes do not stand for actual inheritance, they are only provided to illustrate the applet's logic.

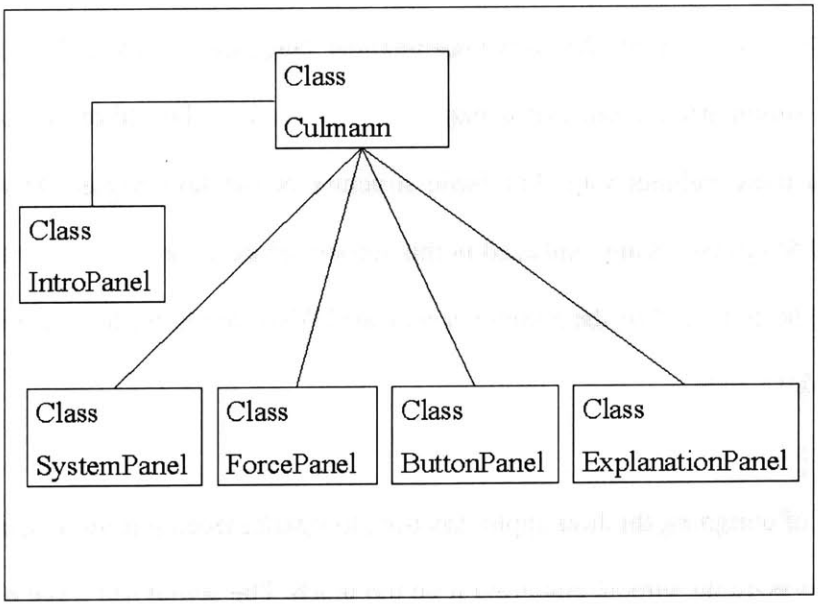

Figure 4-4: Class Diagram of the Culmann applet

Line drawing is understandably a very important part of this applet. In the Java programming language, start and end point define a line. The coordinates of these points are private variables in all the created Java classes, which makes them directly accessible only from the inside of the class where they are defined. In order to make them available to other classes, public methods are given, providing a public interface to the class that can be used to retrieve and modify these variables. The reason for this structure is that it hinders the corruption of data and enhances information hiding.

The most important class of the software product is the class called 'Culmann'. Its content pane holds the four major areas of the applet as described above. It defines the software as an applet by extending the JApplet class of the Java programming language. Besides, this class enables the applet to react to action events, e.g. a mouse click, by implementing the 'ActionListener' interface. All the instances of the various classes that are used in this applet are declared in this class, too. The various actions that have to be performed according to the various options that a user is presented with are defined along with the functions to control the execution of the applet in real-time. Before starting the Culmann method, this class tests the user input given in the text fields and it displays an error message in the text fields if the user has entered invalid data in any of the fields. Depending on a variable that is used to indicate the stage of the applet, this class sets the variables of the corresponding classes concerning the execution of the Culmann method. The mathematical formulas that are needed to transfer this graphical method into a computational program are contained in this class. They are built on the basis of the geometric relationships between the lines that are created during the execution of the Culmann method. A break of thirty milliseconds between every step is implemented to enable the user to follow the method's execution without the need to stop it. On the other hand, if a user wants to stop the execution, he

is also allowed to do that without the need to restart the program from the beginning after the break.

The layout plan of the Culmann method is kept in the so-called 'SystemPanel', an instance of the class of the same name. Initially, the system panel shows the above-mentioned statically determined mechanical system consisting of a beam and two supports with a total of three reaction forces. The system panel provides two different line styles, a solid line style for the outer force and the Culmann line, and a dashed line style for the lines of application. As for all other classes, public methods for private variables are provided that are called by the action events of the 'Culmann' class during execution of the applet to change the appearance of the layout plan. During execution of the program, the outer force is added according to the input given by the user. In the following steps, the lines of application of all forces are drawn and the Culmann line is established. After that, the work in the system panel is done, and the focus shifts to the lower force panel.

The force plan of the Culmann method is displayed in the lower left-hand part of the screen in an object of the class 'ForcePanel'. This panel is empty at the beginning of the applet's execution, only the size and the background color are set. This class mainly consists of private variables to define parallels to the characteristic lines of the layout plan as well as public methods to set these variables dynamically during the applet's run-time, when they are called from within the Culmann class. As in the system panel, both a dashed and a solid line style are provided to visually emphasize the various steps of the Culmann method. The lines in the force plan are only visible at certain stages throughout the software's execution and they must not be visible at certain points of time. The way in which that task is accomplished in this class is similar to the

way it is done in all other classes of this applet. Since a line is characterized in Java code by its start and end point, if the coordinates of these two points match exactly, nothing is drawn. For that reason, the commands to draw the lines are kept alive during the whole life cycle of the applet, but the coordinates of their start and end points are set to zero if the lines are not expected to appear during the current stage of the Culmann method. Similarly, the course of the Culmann method is simulated by sequentially setting the coordinate values for the lines to be drawn. Since the Culmann method only tells us to start with a parallel of the Culmann line in drawing the force plan and does not give us any hints about a reasonable location of a starting point, it was my task to decide where to set the starting point of the force plan. I have chosen to start the Culmann line at the middle of the force plan's left border, and to compute the characteristic coordinates of all other lines, i.e. their start and end points, based on this point.

Especially for educational software, it is desirable to put the user in control of the applet's execution pace. For this reason, I implemented the features that provide these options for a user. The panel that holds these controls is implemented in a class called 'ButtonPanel'. The upper right-hand corner of the main screen shows an object of that class containing the text fields to input the characteristic values of the outer forces as well as the buttons to control the execution of the applet. While the panel is only used to set the foreground and background color of this area, all the buttons, text fields, and labels are defined in the Culmann class. The appearance of this part does not change very much during execution time hence the ButtonPanel class does not contain much code.

The so-called 'ExplanationPanel' in the lower right-hand part of the main screen is used to present the explanation of each step of the Culmann method that is executed simultaneously in

the SystemPanel or in the ForcePanel, respectively. According to a variable that is contained and manipulated dynamically in the Culmann class, the explanatory text that is displayed in this part of the screen is updated to reflect the sequence of the Culmann method. The text is implemented as a row of strings in the Java code. The structure of this code may seem a little bit awkward since a new string is used for every single explanation line, but it enabled me to control the appearance of the text in a way that the text fits the available space in the explanation panel.

Although the Culmann method is explained during execution time on the screen, I added an additional help section about the program itself, in case that a user does not know how to use this program. Aimed at new users, this section is not going to be needed very frequently, hence I decided to exclude it from the main screen and use an additional pop-up window for it. Furthermore, providing a new window for the help section enables the user to keep it visible while trying to explore the program because it is independent of the main screen. The text and executive commands of this help section are part of the applet's main class, the Culmann class.

All the classes described above contain the Java code that is necessary to execute the Culmann method. However, since it is a Java applet, it has to be viewed either embedded in a web page or with a so-called 'appletviewer' that is part of the Java development kit. For this reason, I wrote a very simple HTML file called 'Culmann.html' that enables the running of the software. This file just tells the browser where to find the executable Java code, and it specifies the size of the applet on the screen.

4.3 The Java Applet

This applet is integrated into the educational part of the Flagpole project's web site. It can be found under http://flagpole.mit.edu/applets/culmann/Culmann_plugin_1.3.html.

The following section displays the output of the above-described Java code as well as some possible applications for this software. Besides, future enhancements and extensions are discussed.

After hitting the 'start' button on the introduction screen, the user is taken to the main screen. The main screen holds all information and visualizations necessary to run the applet. For this reason, the basic layout of the main screen does not change during execution time, only its content alters. Although the screen consists of four different areas, it is not cluttered. All buttons, labels, actions and text are clearly visible and easy to grasp. A differentiation of colors further facilitates the understanding of the applet's setup.

The following figure shows the applet's output after completing an example of the Culmann method.

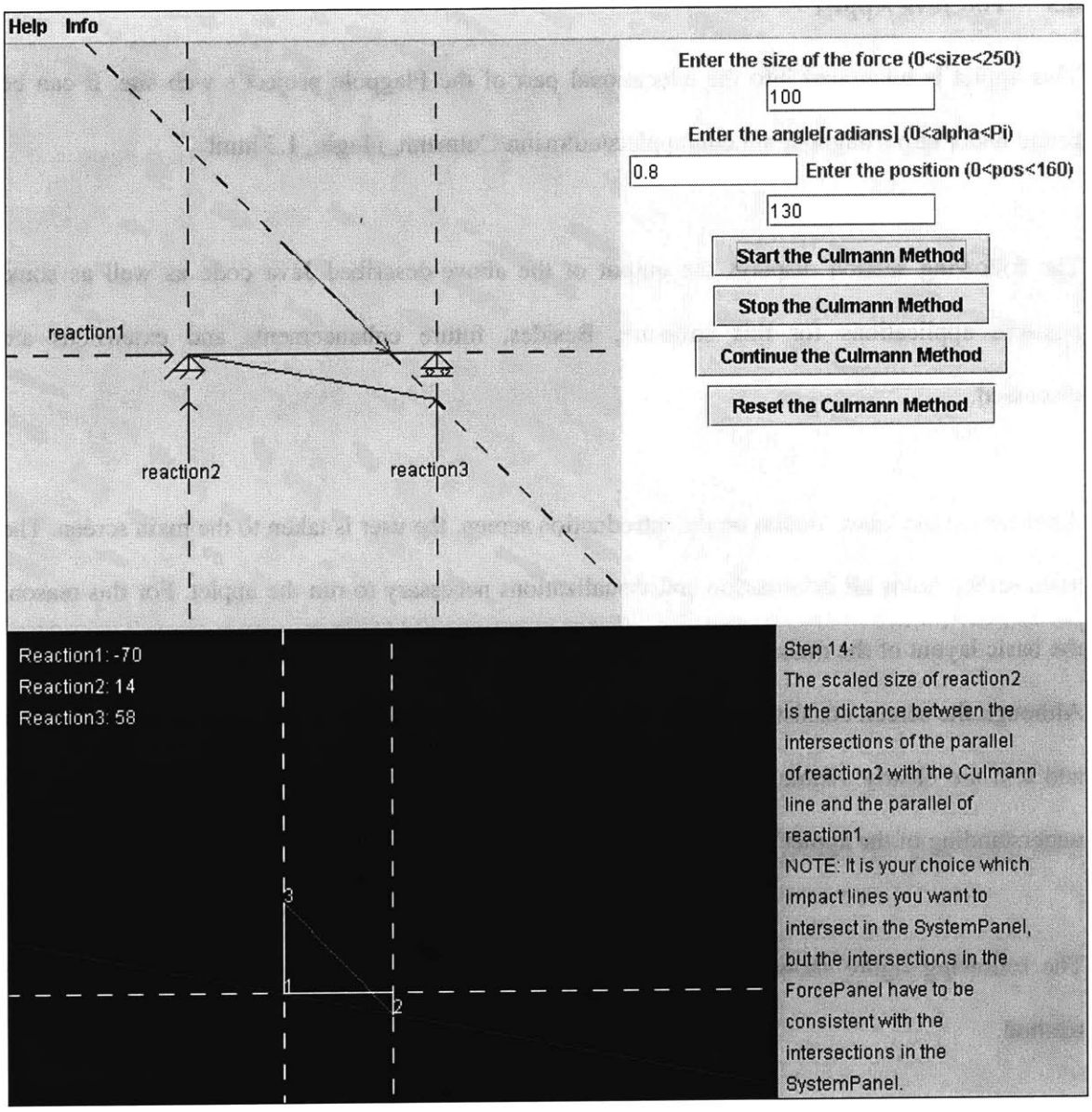

Figure 4-5: The Culmann Method applet's main screen

As mentioned earlier, this software provides a help section that explains how to use this program for new users or users unfamiliar with the concept of the Culmann method. The Culmann method itself is not explained in this section, because it is described on the main screen during execution time of the applet. The figure below shows the help section, contained in a window that is

separated from the main screen, thus enabling the user to keep both of them conveniently available at the same time.

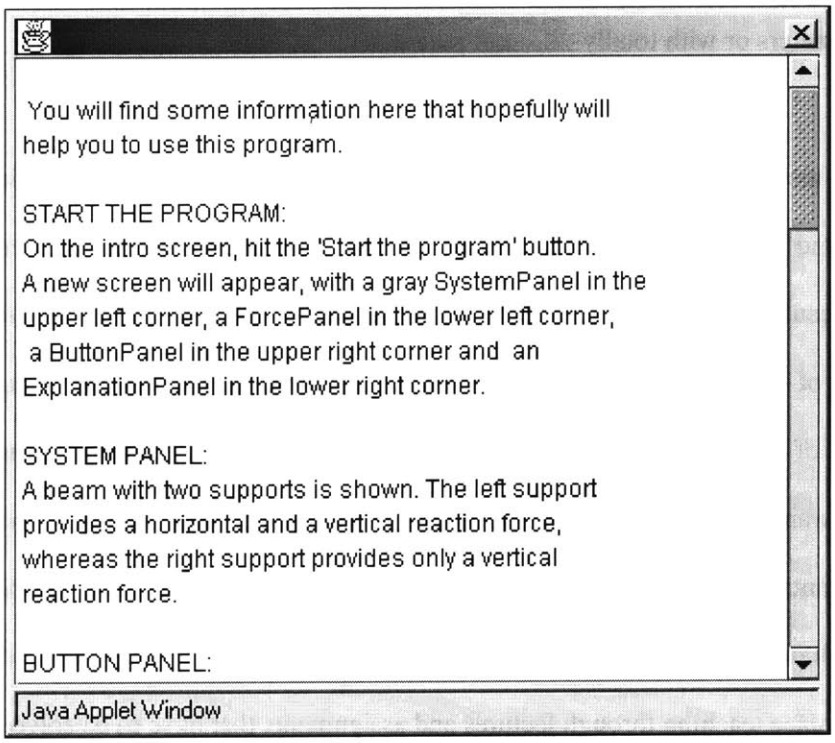

Figure 4-6: The Culmann Method applet's help screen

Before the user can start the applet, the only thing he has to specify is the outer force, which is characterized by its length, its direction, and its position on the system. If any text field is left empty, the applet will not start to run. Similarly, if invalid data is entered in any of the text fields, the user is prompted by the applet to replace the entries with valid data. After starting the execution of the applet, every step of the Culmann method is executed as described above, with the layout plan in the upper part of the screen and the force plan occupying the lower portion as well as an explanation in the lower right hand part of the screen. If the user wants to study a certain step for a longer time, an option to stop and continue the execution is provided.

An option to reset the applet, i.e. to clear the main screen, allows the successive solving of an unlimited number of cases. This feature can be used to investigate various cases with just slightly different parameters or with totally different parameters.

The Culmann method applet provides a learning student with the opportunity to develop a sense for the size and direction of reaction forces. A user can study different system settings and compare the results in order to improve his understanding of the relationship between outer loads and reactions of a statically determined structure. It is suitable for students who have a basic understanding of static forces and equilibrium. However, it can also be useful for more advanced students who want to improve their understanding of basic mechanics or refresh their knowledge. The main advantages of this software are the step-by-step visualization of a mechanical concept and the simultaneous display of the solution development and explanatory notes. It is not intended to replace teaching through lectures and assignments that have to be solved by hand. It is rather supposed to be a supportive tool to these more 'traditional' teaching tools, and its greatest value is probably gained as a companion to a mechanical education.

However, using the applet has shown that there are some things that could be improved in a future version. One future enhancement would be to provide a flexible system setup to the user, i.e. let the user choose how long the beam is, what kinds of supports are implemented, where they are located etc. Increasing the flexibility of the system could also include wider limits for the force parameters. The current applet supports only a limited range of angles and force sizes, which is not a big problem at this stage since the core goal is to explain the Culmann method and not to show every thinkable constellation. One problem that needs to be addressed is the invisibility of the lines in the force plan due to certain system parameters that push them out of

the visible part of the force panel in some cases. Although the applet still executes and the results are correct, it is not helpful if the user cannot see the solution developing. A possible solution to this issue might be the selection of a different starting point in the force panel. Another feature that could add more convenience for a user would be the opportunity not only to stop the applet's execution but also to go back some steps in order to review previous actions.

Future enhancements largely depend on user feedback that could be provided by learning students using this software. Some issues will only arise during the applet's use by people who are not yet familiar with the Culmann method and this program, because they can judge best what would be useful for them in order to improve their learning experience. To get this feedback, it would be very helpful to use this tool in university classes to reach a significant number of users that this applet is intended for. Additional future enhancements and opportunities for the educational software developed for the Flagpole project are discussed in more detail in the next chapter.

5 Conclusion and Outlook

The Flagpole project team has made important achievements during the past months. However, a number of issues still have to be addressed and a large number of future extensions are currently imaginable.

As one would think of a starting research project, we were facing a lot of hurdles and uncertainties in the beginning. We explored techniques that did not prove to be useful for our project in the end. We had to overcome the inexperience of some team members concerning software development in general, especially with the Java programming language. On top of that, using a structure on the MIT campus did not allow us to do much experimenting because of the restrictions given by the administration.

For the decision-support part of the project, the most important short-term goal is to instrument the flagpole on the MIT campus with all the sensors to constantly monitor this structural system. Then, the incoming data have to be analyzed over a continuing and meaningful period of time. This analysis should indicate the maximum loads that appear and the corresponding behavior of the flagpole. On the basis of these results, the flagpole could be equipped to become a 'smart structure', e.g. by dynamically changing the damping parameters in order to enable a real-time response to changing circumstances. Additionally, the monitoring system could indicate the ageing of a structure, e.g. if equal loads cause increasing displacements over time, thus enabling professional observers to guarantee the timely replacement of a failure-prone structure and reducing the risk of potential damage caused by unforeseen system failure. In the long term, that kind of decision-support could be applied not only to a single structure, but also to a whole

system of combined structures. This development could pave the way for the creation of a smart environment, the so-called 'I-City'.

For the educational part of the project, the next step would be to integrate this software into classes as a teaching support tool, and to collect the feedback from teachers and students about its suitability. This has already been partially achieved for the Java applet about the Mohr's circle, which is used in one undergraduate course at MIT. Besides, the existing software does not cover all areas of a mechanical education, i.e. it is necessary to develop additional software to make similar tools available during the whole course of such a class. There are virtually no limits to the applicability of educational software. It can be developed for any field of engineering or science.

Another task of the future is to build 'smart applets', which should be able to generate information about their usage automatically. The idea is to enable the software to register the entry of a user, i.e. to keep track of how a user navigates through the software, how often a certain page is visited, how long it is viewed, etc. The Flagpole team could use this information to analyze the patterns of usage of its educational software. According to the results, the software could be made more efficient or easier to handle. Such a feature should be complementary to the user feedback. It could be misleading if analyzed independently, e.g. the Flagpole team might think that a certain page is not desirable because it has no visits, but in reality the reason for that may be that the users simply do not find this page.

All in all, I think that the Flagpole project team did a good job, producing meaningful results and building a solid basis for future research.

6 Appendix

This appendix contains the code for the Java applet about the Culmann method as explained in chapter 4.

Class Culmann

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.lang.Math;

public class Culmann extends JApplet implements Runnable,ActionListener{
        JMenuBar menuBar;
        JMenu helpMenu, infoMenu;
        JMenuItem helpMI, infoMI;

        IntroPanel introPanel;
        ForcePanel forcePanel;
        ExplanationPanel explanationPanel;
        static SystemPanel systemPanel;
        ButtonPanel buttonPanel;

        Jbutton startProgBut, startBut, startCulBut, stopBut, continueBut, resetBut,
        drawForceBut, drawCulBut;
        JLabel forceLab, angleLab, positionLab;
        JTextField startText, angleText, positionText;
        JPanel contentPane;

        Thread drawThread;
        Thread introThread = null;

        String name;
  boolean frozen = true;
  int i = 1;
  int a = 0;
  int b = 0;
  double c = 0;
  int d = 0;
  int e = 0;
 static double alpha = 0.785398;
```

```
        static double beta = 0;
        static double force_length = 0;
        static double position = 0;
        public static String length1Str = "";
        public static String length2Str = "";
        public static String length3Str = "";
        public static int length1Int = 0;
        public static int length2Int = 0;
        public static int length3Int = 0;
        public static int if_test = 0;
        public static int mover = 0;

        public void init(){
                contentPane = new JPanel();
                        contentPane.setLayout(new BorderLayout());
                setMenuBar();
                        setSystemPanel();
                        setForcePanel();
                        setButtonPanel();
                        setIntroPanel();
                start(0);
                forcePanel.setVisible(false);
                        systemPanel.setVisible(false);
                buttonPanel.setVisible(false);
                        setContentPane(contentPane);
        }

        private void setIntroPanel(){
        introPanel = new IntroPanel();
        introPanel.setMinimumSize(new Dimension(700,800));
        introPanel.setPreferredSize(new Dimension(700,800));
        startProgBut = new JButton ("Start the program");
        startProgBut.addActionListener(this);
        introPanel.add(startProgBut);
        contentPane.add( introPanel, BorderLayout.WEST);
}

private void setMenuBar(){
        menuBar = new JMenuBar();
        helpMenu = new JMenu("Help");
        menuBar.add(helpMenu);
        helpMI = new JMenuItem("Explanation of this program");
        helpMI.addActionListener(this);
        helpMenu.add(helpMI);
        infoMenu = new JMenu("Info");
        menuBar.add(infoMenu);
        infoMI = new JMenuItem("Info about this program");
        infoMI.addActionListener(this);
```

```
        infoMenu.add(infoMI);
        setJMenuBar(menuBar);
}

private void setSystemPanel(){
        systemPanel = new SystemPanel();
        systemPanel.setMinimumSize(new Dimension(400,400));
        systemPanel.setPreferredSize(new Dimension(400,400));
        contentPane.add( systemPanel, BorderLayout.CENTER);
}

private void setForcePanel(){
        forcePanel = new ForcePanel();
        forcePanel.setLayout(new BorderLayout());
        forcePanel.setMinimumSize(new Dimension(500,400));
        forcePanel.setPreferredSize(new Dimension(500,400));
        explanationPanel = new ExplanationPanel();
        explanationPanel.setMinimumSize(new Dimension(200,400));
        explanationPanel.setPreferredSize(new Dimension(200,400));
        forcePanel.add(explanationPanel, BorderLayout.EAST);
        contentPane.add( forcePanel, BorderLayout.SOUTH);
}

 private void setButtonPanel(){
        buttonPanel = new ButtonPanel();
        buttonPanel.setMinimumSize(new Dimension(300,400));
        buttonPanel.setPreferredSize(new Dimension(300,400));
        forceLab = new JLabel("Enter the size of the force (0<size<250)");
        buttonPanel.add(forceLab);
        startText = new JTextField(10);
        startText.addActionListener(this);
        buttonPanel.add(startText);
        angleLab = new JLabel("Enter the angle[radians] (0<alpha<Pi)");
        buttonPanel.add(angleLab);
        angleText = new JTextField(10);
        angleText.addActionListener(this);
        buttonPanel.add(angleText);
        positionLab = new JLabel("Enter the position (0<pos<160)");
        buttonPanel.add(positionLab);
        positionText = new JTextField(10);
        positionText.addActionListener(this);
        buttonPanel.add(positionText);
        startCulBut = new JButton ("Start the Culmann Method");
        startCulBut.addActionListener(this);
        buttonPanel.add(startCulBut);
        stopBut = new JButton ("Stop the Culmann Method");
        stopBut.addActionListener(this);
        buttonPanel.add(stopBut);
```

```
        continueBut = new JButton ("Continue the Culmann Method");
        continueBut.addActionListener(this);
        buttonPanel.add(continueBut);
        resetBut = new JButton ("Reset the Culmann Method");
        resetBut.addActionListener(this);
        buttonPanel.add(resetBut);
        contentPane.add( buttonPanel, BorderLayout.EAST);
}

public void actionPerformed(ActionEvent e){
        Object src = e.getSource();
        if(src == startProgBut){
           systemPanel.setVisible(true);
           introPanel.setVisible(false);
           forcePanel.setVisible(true);
           buttonPanel.setVisible(true);
           menuBar.setVisible(true);
           repaint();
        }
        else if( src == helpMI ){
           JDialog dialog = new JDialog();
           String str = "\n  You will find some information here that hopefully will
                \n help you to use this program. \n \n START THE PROGRAM: \n On the intro
                screen,
                hit the 'Start the program' button. \n A new screen will appear, with a gray
                SystemPanel in the \n upper left corner, a ForcePanel in the lower left corner,
                \n  a ButtonPanel in the upper right corner and  an \n ExplanationPanel
                in the lower right corner. \n \n SYSTEM PANEL: \n A beam with two supports is
                shown.
                The left support \n provides a horizontal and a vertical reaction force,
                \n whereas the right support provides only a vertical \n reaction force.
                \n \n BUTTON PANEL:\n In this panel you'll see all the buttons and text fields
                \n that you need to use this program.\n ENTER THE SIZE OF THE FORCE:
                \n The units don't matter, because the results will  \n depend only on the size
                of the force,\n not on the units.\n ENTER THE ANGLE:\n Enter a value
                between 0 and 3.14159. \n If your angle is greater than that,\n just subtract
                3.14159
                from it and switch the sign\n of the results from 'minus' to 'plus' and vice versa.
                \n ENTER THE POSITION:\n Since the size of the beam is fixed here (160),
                \n you can calculate the corresponding position of your system,\n e.g. if your
                beam
                is 6 meters long and the force\n is located at 3 meters, just enter '80' here.
                \n START THE CULMANN METHOD:\n Once you've entered valid numbers in
                every field,
                \n you can start the program with this button.\n If you entered invalid numbers
                in any field, you will\n be prompted to re-enter all numbers.\n
                STOP THE CULMANN METHOD:\n By clicking on this button, you can stop
                \n the program at anytime, e.g. to take e longer look\n at the explanations.
```

```
            \n CONTINUE THE CULMANN METHOD:\n This button allows you to
            continue \n your program
            after you stopped it.\n RESET THE CULMANN METHOD:\n If you want to
            enter a new system,
            click on this button\n and all the entries will be cleared.\n \n FORCE PANEL:
            \n The graphical solution of the problem will be shown here.\n Look at the
            ExplanationPanel to see what is happ
            ening.\n DISPLAY OF THE REACTION FORCES: \n A 'minus' before a result
            means
            that the direction \n of that force is opposite to its direction\n as it is displayed
            in the SystemPanel! \n RESULTS: \n It might happen that the drawing of the
            forces
            \n in the ForcePanel are not visible inside the displayed area. \n Nevertheless,
            the program will display the correct results \n for the forces!
            \n \n EXPLANATION PANEL:\n You will find explanations here,
            corresponding to
            \n every step that the program is currently executing. \n \n THANKS FOR
            USING
            THIS PROGRAM, I HOPE IT WILL HELP! \n \n";
        JTextArea helpText = new JTextArea (str);
        helpText.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(helpText);
        scrollPane.setPreferredSize(new Dimension(400,200));
        dialog.getContentPane().add(scrollPane, BorderLayout.CENTER);
        dialog.pack();
        dialog.setVisible(true);
    }
    else if(src == infoMI){
        JDialog dialog = new JDialog();
        String str = "This is a program about the Culmann Method, \na graphical method for
        solving beam reaction forces. \nIt is very well suitable for the learning student \nto
        better his/her understanding of mechanical principles.\n";
        JTextArea infoText = new JTextArea (str);
        infoText.setEditable(false);
        infoText.setPreferredSize(new Dimension(400,80));
        dialog.getContentPane().add(infoText, BorderLayout.CENTER);
        JLabel copyrightlabel = new JLabel("Copyright 2000 by Jochen Schlingloff");
        dialog.getContentPane().add(copyrightlabel, BorderLayout.SOUTH);
        dialog.pack();
        dialog.setVisible(true);
    }
    else if(src == startCulBut)
        startCulmann();
    else if(src == stopBut){
        stopCulmann();
    }
    else if(src == continueBut){
        continueCulmann();
```

```
        }
        else if(src == resetBut){
            resetCulmann();
            repaint();
        }
}

public static SystemPanel getSystemPanel(){
        return systemPanel;
}

public void startCulmann(){
        frozen = false;
        start(1);
}

public void stopCulmann(){
        frozen = true;
        drawThread = null;
}

public void continueCulmann(){
        frozen = false;
        start(1);
}

public void resetCulmann(){
        frozen = true;
        if_test = 0;
        i = 1;
        systemPanel.setForce(0,0,0,0);
        systemPanel.setArr1(0,0,0,0);
        systemPanel.setArr2(0,0,0,0);
        systemPanel.setForce_Line(0,0,0,0);
        systemPanel.setCul(0,0,0,0);
        systemPanel.setReac1(0,0,0,0);
        systemPanel.setReac2(0,0,0,0);
        systemPanel.setReac3(0,0,0,0);
        forcePanel.set_par_Cul(0,0,0,0);
        forcePanel.set_par_Force(0,0,0,0);
        forcePanel.set_par_Reac3_Line(0,0,0,0);
        forcePanel.set_par_Reac3(0,0,0,0);
        forcePanel.set_par_Reac1_Line(0,0,0,0);
        forcePanel.set_par_Reac1(0,0,0,0);
        forcePanel.set_par_Reac2_Line(0,0,0,0);
        forcePanel.set_par_Reac2(0,0,0,0);
        angleText.setText("0");
        startText.setText("0");
```

```
        positionText.setText("0");
        forcePanel.str1 = "";
        forcePanel.str2 = "";
        forcePanel.str3 = "";
        length1Str = "";
        length2Str = "";
        length3Str = "";
        length1Int = 0;
        length2Int = 0;
        length3Int = 0;
        drawThread = null;
}

public void start(int z){
        if (z==0){
          introThread = new Thread(this);
          introThread.start();
        }
        else if((!frozen) && (z==1)){
          if(drawThread == null){
                drawThread = new Thread(this);
          }
          drawThread.start();
        }
}

public void run(){
        if (Thread.currentThread() == introThread) {
          while(mover < 200){
                repaint();
                mover += 1;
                try{
                  Thread.sleep(20);
                }
                catch(InterruptedException exc_1){
                  break;
                }
          }
        }
        else if (Thread.currentThread() == drawThread){
          for  ( ; (Thread.currentThread() == drawThread) &&  (i < 720) ; i+= 1){
                //get the values from the text fields!!!
                alpha = Double.parseDouble(angleText.getText());
                force_length = Double.parseDouble(startText.getText());
                position = Double.parseDouble(positionText.getText());
                // test, if the user has entered valid data
                if( (alpha<=0) || (alpha>=3.1415927) || (force_length<=0) || (force_length>250) ||
                (position<0) || (position>160) ){
```

```
    startText.setText("ENTER VALID DATA");
    angleText.setText("ENTER VALID DATA");
    positionText.setText("ENTER VALID DATA");
    break;
}
if ( i ==1 ){
    if_test = 1;
    //draw the force onto the SystemPanel
    a = (int)((SystemPanel.get_w())*3/10 + position);
    b = (int)((SystemPanel.get_w())*3/10 + position -
(Math.cos(alpha))*force_length);
    d = (int)(SystemPanel.get_h()/2);
    e = (int)(SystemPanel.get_h()/2-Math.sin(alpha)*force_length);
    systemPanel.setForce(a,d,b,e);
    //make the force line an arrow
    if(alpha <= 1.5707963){
    systemPanel.setArr1(a,d,a-(int)(Math.cos(alpha/2)*10),d-
(int)(Math.sin(alpha/2)*10));
    systemPanel.setArr2(a,d,a-(int)(Math.sin((1.5707963-alpha)/2)*10),d-
(int)(Math.cos((1.5707963-alpha)/2)*10));
    }
    else  if(alpha > 1.5707963){
    systemPanel.setArr1(a,d,a+(int)(Math.cos((3.1415927-alpha)/2)*10),d-
(int)(Math.sin((3.1415927-alpha)/2)*10));
    systemPanel.setArr2(a,d,a-(int)(Math.sin((1.5707963-alpha)/2)*10),d-
(int)(Math.cos((1.5707963-alpha)/2)*10));
    }
}
else if ( i== 100 ){
    if_test = 2;
    int a1 = 0;
    int b1 = SystemPanel.get_h()/2;
    int c1 = SystemPanel.get_w();
    int d1 = SystemPanel.get_h()/2;
    systemPanel.setReac1(a1,b1,c1,d1);
}
else if ( i==150 ){
    if_test = 3;
    int a2 = SystemPanel.get_w()*3/10;
    int b2 = 0;
    int c2 = SystemPanel.get_w()*3/10;
    int d2 = SystemPanel.get_h();
    systemPanel.setReac2(a2,b2,c2,d2);
}
else if ( i==200 ){
    if_test = 4;
    int a3 = SystemPanel.get_w()*7/10;
    int b3 = 0;
```

```
   int c3 = SystemPanel.get_w()*7/10;
   int d3 = SystemPanel.get_h();
   systemPanel.setReac3(a3,b3,c3,d3);
}
else if ( i==250 ){
   if_test = 5;
   a = (int)((SystemPanel.get_w())*3/10 + position -
SystemPanel.get_h()/2/(Math.tan(alpha)));
   b = (int)((SystemPanel.get_w())*3/10 + position +
SystemPanel.get_h()/2/(Math.tan(alpha)));
   systemPanel.setForce_Line(a,0,b,(SystemPanel.get_h()));
   }
else if ( i==300 ){
   if_test = 6;
   c = ( Math.tan(alpha)*(SystemPanel.get_w()*4/10-position) );
   int cdraw = (int)c;

systemPanel.setCul(((SystemPanel.get_w())*3/10),((SystemPanel.get_h())/2),((S
ystemPanel.get_w())*7/10),(SystemPanel.get_h()/2+cdraw));
}
else if ( i==350 ){
   if_test = 7;
   beta = Math.atan(c/(SystemPanel.get_w()*4/10));
   int betadraw = (int)beta;
   int a4 = 0;
   int b4 = forcePanel.height/2;
   int c4 = forcePanel.width;
   int d4 = (int)(forcePanel.height/2+(forcePanel.width*Math.tan(beta)));
   forcePanel.set_par_Cul(a4,b4,c4,d4);
}
else if ( i==400 ){
   if_test = 8;
   int a5 = (int)(forcePanel.width/2);
   int b5 = (int)(forcePanel.height/2+(forcePanel.width/2*Math.tan(beta)));
   int c5 = (int)(forcePanel.width/2-Math.cos(alpha)*force_length);
   int d5 = (int)(forcePanel.height/2+(forcePanel.width/2*Math.tan(beta)-
Math.sin(alpha)*force_length));
   forcePanel.set_par_Force(a5,b5,c5,d5);
}
else if ( i==450 ){
   if_test = 9;
   int a6 = (int)(forcePanel.width/2-Math.cos(alpha)*force_length);
   int b6 = 0;
   int c6 = (int)(forcePanel.width/2-Math.cos(alpha)*force_length);
   int d6 = forcePanel.height;
   forcePanel.set_par_Reac3_Line(a6,b6,c6,d6);
}
else if ( i== 500 ){
```

```
    if_test = 10;
    int a7 = 0;
    int b7 = (int)(forcePanel.height/2+(forcePanel.width/2*Math.tan(beta)-
Math.tan(beta)*Math.cos(alpha)*force_length));
    int c7 = forcePanel.width;
    int d7 = (int)(forcePanel.height/2+forcePanel.width/2*Math.tan(beta)-
Math.tan(beta)*Math.cos(alpha)*force_length);
    forcePanel.set_par_Reac1_Line(a7,b7,c7,d7);
  }
  else if ( i==550 ){
    if_test = 11;
    int a8 = (int)(forcePanel.width/2);
    int b8 = (int)(forcePanel.height);
    int c8 = (int)(forcePanel.width/2);
    int d8 = 0;
    forcePanel.set_par_Reac2_Line(a8,b8,c8,d8);
  }
  else if ( i==600 ){
    if_test = 12;
    int a9 = (int)(forcePanel.width/2-Math.cos(alpha)*force_length);
    int b9 = (int)(forcePanel.height/2+(forcePanel.width/2*Math.tan(beta)-
Math.sin(alpha)*force_length));
    int c9 = (int)(forcePanel.width/2-Math.cos(alpha)*force_length);
    int d9 = (int)(forcePanel.height/2+forcePanel.width/2*Math.tan(beta)-
Math.tan(beta)*Math.cos(alpha)*force_length);
    forcePanel.set_par_Reac3(a9,b9,c9,d9);
    length3Int = d9 - b9;
  }
  else if ( i==650 ){
    if_test = 13;
    int a10 = (int)(forcePanel.width/2-(Math.cos(alpha)*force_length));
    int b10 = (int)(forcePanel.height/2+forcePanel.width/2*Math.tan(beta)-
Math.tan(beta)*Math.cos(alpha)*force_length);
    int c10 = (int)(forcePanel.width/2);
    int d10 = (int)(forcePanel.height/2+forcePanel.width/2*Math.tan(beta)-
Math.tan(beta)*Math.cos(alpha)*force_length);
    forcePanel.set_par_Reac1(a10,b10,c10,d10);
    length1Int = a10 - c10;
  }
  else if ( i==700 ){
    if_test = 14;
    int a11 = (int)(forcePanel.width/2);
    int b11 = (int)(forcePanel.height/2+(forcePanel.width/2*Math.tan(beta)));
    int c11 = (int)(forcePanel.width/2);
    int d11 = (int)(forcePanel.height/2+forcePanel.width/2*Math.tan(beta)-
(Math.tan(beta)*Math.cos(alpha)*force_length));
    forcePanel.set_par_Reac2(a11,b11,c11,d11);
    length2Int = b11 - d11;
```

```
                    forcePanel.str1 = "1";
                    forcePanel.str2 = "2";
                    forcePanel.str3 = "3";
                    length1Str = "Reaction1: " + length1Int;
                    length2Str = "Reaction2: " + length2Int;
                    length3Str = "Reaction3: " + length3Int;
                }
                try{
                    Thread.sleep(30);
                }
                catch(InterruptedException exc_2){
                    break;
                }
                repaint();
            }
        }
        drawThread = null;
    }

    public static double get_beta(){
        return beta;
    }

    public static double get_alpha(){
        return alpha;
    }

    public static double get_force_length(){
        return force_length;
    }
}
```

Class IntroPanel

```
import javax.swing.*;
import java.awt.*;

public class IntroPanel extends JPanel{

    public IntroPanel(){
        super();
        setBackground(Color.black);
        setForeground(Color.white);
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
```

```
        Graphics2D g2 = (Graphics2D)g;
        g2.drawString("The Culmann Method",(Culmann.mover+35),200);
        g2.drawString("by",290,250);
        g2.drawString("Jochen Schlingloff",(450-Culmann.mover),300);
    }
}
```

Class SystemPanel

```
import javax.swing.*;
import java.awt.*;

public class SystemPanel extends JPanel{
    static int w = 400;
    static int h = 400;
    int reac1_x1 = 0; //reac1: the horizontal reaction force
    int reac1_y1 = 0;
    int reac1_x2 = 0;
    int reac1_y2 = 0;
    int reac2_x1 = 0; //reac2: the left vertical reacton force
    int reac2_y1 = 0;
    int reac2_x2 = 0;
    int reac2_y2 = 0;
    int reac3_x1 = 0; //reac3: the right vertical reaction force
    int reac3_y1 = 0;
    int reac3_x2 = 0;
    int reac3_y2 = 0;
    int force_line_x1 = 0; //force: the external force
    int force_line_y1 = 0;
    int force_line_x2 = 0;
    int force_line_y2 = 0;
    int force_x1 = 0; //force: the external force
    int force_y1 = 0;
    int force_x2 = 0;
    int force_y2 = 0;
    int cul_x1 = 0; //cul: the points of the Culmann line
    int cul_y1 = 0;
    int cul_x2 = 0;
    int cul_y2 = 0;
    int arr1_x1 = 0;
    int arr1_y1 = 0;
    int arr1_x2 = 0;
    int arr1_y2 = 0;
    int arr2_x1 = 0;
    int arr2_y1 = 0;
    int arr2_x2 = 0;
    int arr2_y2 = 0;
```

```
  final static float dash[] = {10.0f};
  final static BasicStroke dashedStroke = new BasicStroke(1.0f,
BasicStroke.CAP_BUTT,BasicStroke.JOIN_MITER, 10.0f, dash, 0.0f);
  final static BasicStroke solidStroke = new BasicStroke(1.0f);
  BasicStroke stroke = solidStroke;

  public SystemPanel(){
      super();
      setBackground(Color.lightGray);
      setForeground(Color.black);
  }

  void setStroke(BasicStroke s){
      stroke = s;
      repaint();
  }

  void setColor(Color color){
      setForeground(color);
      repaint();
  }

  public void paintComponent(Graphics g){
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;

      //code for the initial display of the system
      g2.drawLine(w*3/10,h/2,w*7/10,h/2); //the line for the beam
      g2.drawLine(w*3/10-10,h/2+10,w*3/10,h/2);//line for the support
      g2.drawLine(w*3/10+10,h/2+10,w*3/10,h/2);//line for the supprt
      g2.drawLine(w*3/10-10,h/2+10,w*3/10+10,h/2+10);//horizontal supp. line
      g2.drawLine(w*3/10-10,h/2+10,w*3/10-15,h/2+15);
      g2.drawLine(w*3/10,h/2+10,w*3/10-5,h/2+15);
      g2.drawLine(w*3/10+10,h/2+10,w*3/10+5,h/2+15);
      g2.drawLine(w*7/10-10,h/2+10,w*7/10,h/2);
      g2.drawLine(w*7/10+10,h/2+10,w*7/10,h/2);
      g2.drawLine(w*7/10-10,h/2+10,w*7/10+10,h/2+10);
      g2.drawLine(w*7/10-10,h/2+15,w*7/10+10,h/2+15);
      g2.drawOval(w*7/10-7,h/2+10,5,5);
      g2.drawOval(w*7/10+3,h/2+10,5,5);
      g2.drawLine(w*3/10-40,h/2,w*3/10-10,h/2);//reaction1
      g2.drawLine(w*3/10-10,h/2,w*3/10-15,h/2-5);
      g2.drawLine(w*3/10-10,h/2,w*3/10-15,h/2+5);
      g2.drawString("reaction1",w*3/10-90,h/2-10);
      g2.drawLine(w*3/10,h/2+60,w*3/10,h/2+30);//reaction2
      g2.drawLine(w*3/10,h/2+30,w*3/10-5,h/2+35);
      g2.drawLine(w*3/10,h/2+30,w*3/10+5,h/2+35);
```

```
        g2.drawString("reaction2",w*3/10-30,h/2+80);
        g2.drawLine(w*7/10,h/2+60,w*7/10,h/2+30);//reaction3
        g2.drawLine(w*7/10,h/2+30,w*7/10-5,h/2+35);
        g2.drawLine(w*7/10,h/2+30,w*7/10+5,h/2+35);
        g2.drawString("reaction3",w*7/10-30,h/2+80);

        //set the stroke here ("dashed") and draw the lines of application
        g2.setStroke(dashedStroke);

        g2.drawLine(reac1_x1,reac1_y1,reac1_x2,reac1_y2);
        g2.drawLine(reac2_x1,reac2_y1,reac2_x2,reac2_y2);
        g2.drawLine(reac3_x1,reac3_y1,reac3_x2,reac3_y2);

        g2.setColor(Color.red);
        g2.setStroke(solidStroke);

        g2.drawLine(force_x1,force_y1,force_x2,force_y2);
        g2.drawLine(arr1_x1,arr1_y1,arr1_x2,arr1_y2);
        g2.drawLine(arr2_x1,arr2_y1,arr2_x2,arr2_y2);

        g2.setColor(Color.black);
        g2.setStroke(dashedStroke);

        g2.drawLine(force_line_x1,force_line_y1,force_line_x2,force_line_y2);

        g2.setStroke(solidStroke);

        g2.setColor(Color.blue);
        g2.drawLine(cul_x1,cul_y1,cul_x2,cul_y2);
    }

    public void setForce(int a, int b, int c, int d){
        force_x1 = a;
        force_y1 = b;
        force_x2 = c;
        force_y2 = d;
    }

    public void setArr1(int a, int b, int c, int d){
        arr1_x1 = a;
        arr1_y1 = b;
        arr1_x2 = c;
        arr1_y2 = d;
    }

    public void setArr2(int a, int b, int c, int d){
        arr2_x1 = a;
        arr2_y1 = b;
```

```
        arr2_x2 = c;
        arr2_y2 = d;
}

public void setForce_Line(int a, int b, int c, int d){
        force_line_x1 = a;
        force_line_y1 = b;
        force_line_x2 = c;
        force_line_y2 = d;
}

 public void setCul(int r, int s, int t, int u){
        cul_x1 = r;
        cul_y1 = s;
        cul_x2 = t;
        cul_y2 = u;
}

public void setReac1(int r, int s, int t, int u){
        reac1_x1 = r; //reac1: the horizontal reaction force
        reac1_y1 = s;
        reac1_x2 = t;
        reac1_y2 = u;
}

public void setReac2(int r, int s, int t, int u){
        reac2_x1 = r; //reac2: the left vertical reaction force
        reac2_y1 = s;
        reac2_x2 = t;
        reac2_y2 = u;
}

public void setReac3(int r, int s, int t, int u){
        reac3_x1 = r; //reac3: the right vertical reaction force
        reac3_y1 = s;
        reac3_x2 = t;
        reac3_y2 = u;
}


 public int get_cul_x1(){
        return cul_x1;
}

 public int get_cul_y1(){
        return cul_y1;
}
```

```
    public int get_cul_x2(){
        return cul_x2;
    }

    public int get_cul_y2(){
        return cul_y2;
    }

     public static int get_w(){
        return w;
    }

     public static int get_h(){
        return h;
        }
}
```

Class ForcePanel

```
import javax.swing.*;
import java.awt.*;
import java.lang.Math;

public class ForcePanel extends JPanel{
    public int width = 500;
    public int height = 400;
    int par_cul_x1 = 0;
    int par_cul_y1 = 0;
    int par_cul_x2 = 0;
    int par_cul_y2 = 0;
    int par_force_x1 = 0;
    int par_force_y1 = 0;
    int par_force_x2 = 0;
    int par_force_y2 = 0;
    int par_reac1_line_x1 = 0;
    int par_reac1_line_y1 = 0;
    int par_reac1_line_x2 = 0;
    int par_reac1_line_y2 = 0;
    int par_reac2_line_x1 = 0;
    int par_reac2_line_y1 = 0;
    int par_reac2_line_x2 = 0;
    int par_reac2_line_y2 = 0;
    int par_reac3_line_x1 = 0;
    int par_reac3_line_y1 = 0;
    int par_reac3_line_x2 = 0;
    int par_reac3_line_y2 = 0;
    int par_reac1_x1 = 0;
```

```
int par_reac1_y1 = 0;
int par_reac1_x2 = 0;
int par_reac1_y2 = 0;
int par_reac2_x1 = 0;
int par_reac2_y1 = 0;
int par_reac2_x2 = 0;
int par_reac2_y2 = 0;
int par_reac3_x1 = 0;
int par_reac3_y1 = 0;
int par_reac3_x2 = 0;
int par_reac3_y2 = 0;
public String str1 = "";
public String str2 = "";
public String str3 = "";

final static float dash[] = {10.0f};
final static BasicStroke dashedStroke = new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
      BasicStroke.JOIN_MITER, 10.0f, dash, 0.0f);
final static BasicStroke solidStroke = new BasicStroke(1.0f);
BasicStroke stroke = solidStroke;

public ForcePanel(){
      super();
      setBackground(Color.black);
      setForeground(Color.white);
}

void setStroke(BasicStroke s){
      stroke = s;
      repaint();
}

public void paintComponent(Graphics g){
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;

      g2.setColor(Color.blue);
      g2.drawLine(par_cul_x1,par_cul_y1,par_cul_x2,par_cul_y2);

      g2.setColor(Color.red);
      g2.drawLine(par_force_x1,par_force_y1,par_force_x2,par_force_y2);

      g2.setColor(Color.white);
      g2.setStroke(dashedStroke);

      g2.drawLine(par_reac1_line_x1,par_reac1_line_y1,par_reac1_line_x2,par_reac1_line_y2
);
```

```
        g2.drawLine(par_reac2_line_x1,par_reac2_line_y1,par_reac2_line_x2,par_reac2_line_y2
);
        g2.drawLine(par_reac3_line_x1,par_reac3_line_y1,par_reac3_line_x2,par_reac3_line_y2
);

        g2.setColor(Color.yellow);
        g2.setStroke(solidStroke);

        g2.drawLine(par_reac1_x1,par_reac1_y1,par_reac1_x2,par_reac1_y2);
        g2.drawLine(par_reac2_x1,par_reac2_y1,par_reac2_x2,par_reac2_y2);
        g2.drawLine(par_reac3_x1,par_reac3_y1,par_reac3_x2,par_reac3_y2);

        g2.drawString(str1,par_reac1_x1,par_reac1_y1);
        g2.drawString(str2,par_reac2_x1,par_reac2_y1);
        g2.drawString(str3,par_reac3_x1,par_reac3_y1);

        g2.drawString(Culmann.length1Str,10,20);
        g2.drawString(Culmann.length2Str,10,40);
        g2.drawString(Culmann.length3Str,10,60);
    }

    public void set_par_Cul(int r, int s, int t, int u){
        par_cul_x1 = r;
        par_cul_y1 = s;
        par_cul_x2 = t;
        par_cul_y2 = u;
    }

    public void set_par_Force(int r, int s, int t, int u){
        par_force_x1 = r;
        par_force_y1 = s;
        par_force_x2 = t;
        par_force_y2 = u;
    }

    public void set_par_Reac1_Line(int r, int s, int t, int u){
        par_reac1_line_x1 = r;
        par_reac1_line_y1 = s;
        par_reac1_line_x2 = t;
        par_reac1_line_y2 = u;
    }

    public void set_par_Reac1(int r, int s, int t, int u){
        par_reac1_x1 = r;
        par_reac1_y1 = s;
        par_reac1_x2 = t;
        par_reac1_y2 = u;
    }
```

```
    public void set_par_Reac2_Line(int r, int s, int t, int u){
        par_reac2_line_x1 = r;
        par_reac2_line_y1 = s;
        par_reac2_line_x2 = t;
        par_reac2_line_y2 = u;
    }

    public void set_par_Reac2(int r, int s, int t, int u){
        par_reac2_x1 = r;
        par_reac2_y1 = s;
        par_reac2_x2 = t;
        par_reac2_y2 = u;
    }

    public void set_par_Reac3_Line(){
        par_reac3_line_x1 = (int)(width/2-
Math.cos(Culmann.get_alpha())*Culmann.get_force_length());
        par_reac3_line_y1 = 0;
        par_reac3_line_x2 = (int)(width/2-
Math.cos(Culmann.get_alpha())*Culmann.get_force_length());
        par_reac3_line_y2 = (int)(height);
    }

    public void set_par_Reac3_Line(int r, int s, int t, int u){
        par_reac3_line_x1 = r;
        par_reac3_line_y1 = s;
        par_reac3_line_x2 = t;
        par_reac3_line_y2 = u;
    }

    public void set_par_Reac3(int r, int s, int t, int u){
        par_reac3_x1 = r;
        par_reac3_y1 = s;
        par_reac3_x2 = t;
        par_reac3_y2 = u;
    }
}
```

Class ButtonPanel

```
import javax.swing.*;
import java.awt.*;

public class ButtonPanel extends JPanel{

    public ButtonPanel(){
```

```
        super();
        setBackground(Color.yellow);
        setForeground(Color.white);
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
    }
}
```

Class ExplanationPanel

```
import javax.swing.*;
import java.awt.*;

public class ExplanationPanel extends JPanel{
    int xstart_par = 0;
    int ystart_par = 0;
    int xend_par = 0;
    int yend_par = 0;

    public ExplanationPanel(){
        super();
        setBackground(Color.cyan);
        setForeground(Color.black);
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;

        if (Culmann.if_test == 0){
            g2.drawString("The program has not ",5 ,20);
            g2.drawString("been started yet.",5 ,40);
            g2.drawString("Please enter the size,",5 ,60);
            g2.drawString("the angle and the position",5 ,80);
            g2.drawString("of the force.",5 ,100);
        }
        else if (Culmann.if_test == 1){
            g2.drawString("Step 1:",5 ,20);
            g2.drawString("The force is displayed",5 ,40);
            g2.drawString("on the system at the entered",5 ,60);
            g2.drawString("position, with the corresponding",5 ,80);
            g2.drawString("size and angle.",5 ,100);
        }
        else if (Culmann.if_test == 2){
            g2.drawString("Step 2:",5 ,20);
```

```
    g2.drawString("The imapct line of reaction1",5 ,40);
    g2.drawString("(the horizontal reaction force)",5 ,60);
    g2.drawString("is drawn.",5 ,80);
  }
  else if (Culmann.if_test == 3){
    g2.drawString("Step 3:",5 ,20);
    g2.drawString("The impact line of reaction2",5 ,40);
    g2.drawString("(the vertical reaction force",5 ,60);
    g2.drawString("on the left side)",5 ,80);
    g2.drawString("is drawn.",5 ,100);
  }
  else if (Culmann.if_test == 4){
    g2.drawString("Step 4:",5 ,20);
    g2.drawString("The impact line of reaction3",5 ,40);
    g2.drawString("(the vertical reaction force",5 ,60);
    g2.drawString("on the right side)",5 ,80);
    g2.drawString("is drawn.",5 ,100);
  }
  else if (Culmann.if_test == 5){
    g2.drawString("Step 5:",5 ,20);
    g2.drawString("The impact line of the ",5 ,40);
    g2.drawString("outer force is drawn.",5 ,60);
  }
  else if (Culmann.if_test == 6){
    g2.drawString("Step 6:",5 ,20);
    g2.drawString("The Culmann line is drawn.",5 ,40);
    g2.drawString("It connects the intersection",5 ,60);
    g2.drawString("of the impact lines of",5 ,80);
    g2.drawString("reaction1 and reaction2 ",5 ,100);
    g2.drawString("with the intersection",5 ,120);
    g2.drawString("of the impact lines of",5 ,140);
    g2.drawString("reaction3 and the outer force.",5 ,160);
  }
  else if (Culmann.if_test == 7){
    g2.drawString("Step 7:",5 ,20);
    g2.drawString("A parallel of the Culmann line",5 ,40);
    g2.drawString("in the SystemPanel",5 ,60);
    g2.drawString("is drawn on the ForcePanel.",5 ,80);
  }
  else if (Culmann.if_test == 8){
    g2.drawString("Step 8:",5 ,20);
    g2.drawString("A scaled  parallel of",5 ,40);
    g2.drawString("the outer force",5 ,60);
    g2.drawString("is drawn in the ForcePanel",5 ,80);
    g2.drawString("on a randomly chosen point.",5 ,100);
    g2.drawString("Only requirement:",5 ,120);
    g2.drawString("the end point of the force",5 ,140);
    g2.drawString("has to be on the Culmann line.",5 ,160);
```

```
}
else if (Culmann.if_test == 9){
   g2.drawString("Step 9:",5 ,20);
   g2.drawString("A parallel of the impact line",5 ,40);
   g2.drawString("of reaction3 is drawn at the",5 ,60);
   g2.drawString("'free' end of the outer force,",5 ,80);
   g2.drawString("i.e. the end that does not",5 ,100);
   g2.drawString("lie on the Culmann line.",5 ,120);
}
else if (Culmann.if_test == 10){
   g2.drawString("Step 10:",5 ,20);
   g2.drawString("A parallel of the impact line",5 ,40);
   g2.drawString("of reaction1 is drawn at the",5 ,60);
   g2.drawString("intersection of the Culmann",5 ,80);
   g2.drawString("line and the parallel of ",5 ,100);
   g2.drawString("the impact line of reaction3.",5 ,120);
}
else if (Culmann.if_test == 11){
   g2.drawString("Step 11:",5 ,20);
   g2.drawString("A parallel of the impact line",5 ,40);
   g2.drawString("of reaction2 is drawn at the",5 ,60);
   g2.drawString("point where the parallel",5 ,80);
   g2.drawString("of the outer force",5 ,100);
   g2.drawString("intersects the Culmann line.",5 ,120);
}
else if (Culmann.if_test == 12){
   g2.drawString("Step 12:",5 ,20);
   g2.drawString("A graphical equilibrium has",5 ,40);
   g2.drawString("been created.",5 ,60);
   g2.drawString("The scaled size of reaction3",5 ,80);
   g2.drawString("is the dictance between the",5 ,100);
   g2.drawString("intersections of the parallel",5 ,120);
   g2.drawString("of reaction3 with the Culmann",5 ,140);
   g2.drawString("line and the parallel of",5 ,160);
   g2.drawString("the outer force.",5 ,180);
}
else if (Culmann.if_test == 13){
   g2.drawString("Step 13:",5 ,20);
   g2.drawString("The scaled size of reaction1",5 ,40);
   g2.drawString("is the dictance between the",5 ,60);
   g2.drawString("intersections of the parallel",5 ,80);
   g2.drawString("of reaction1 with the Culmann",5 ,100);
   g2.drawString("line and the parallel of",5 ,120);
   g2.drawString("reaction2.",5 ,140);
}
else if (Culmann.if_test == 14){
   g2.drawString("Step 14:",5 ,20);
   g2.drawString("The scaled size of reaction2",5 ,40);
```

```
            g2.drawString("is the dictance between the",5 ,60);
            g2.drawString("intersections of the parallel",5 ,80);
            g2.drawString("of reaction2 with the Culmann",5 ,100);
            g2.drawString("line and the parallel of",5 ,120);
            g2.drawString("reaction1.",5 ,140);
            g2.drawString("NOTE: It is your choice which",5 ,160);
            g2.drawString("impact lines you want to",5 ,180);
            g2.drawString("intersect in the SystemPanel,",5 ,200);
            g2.drawString("but the intersections in the",5 ,220);
            g2.drawString("ForcePanel have to be",5 ,240);
            g2.drawString("consistent with the",5 ,260);
            g2.drawString("intersections in the",5 ,280);
            g2.drawString("SystemPanel.",5,300);
            g2.drawString("THIS IS THE END!",5 ,320);
        }

    }
}
```

Culmann.html

```
<HTML>
<HEAD>
<TITLE> The Culmann Method </TITLE>
</HEAD>

<BODY>
<APPLET CODE="Culmann.class" WIDTH=700 HEIGHT=800>
</APPLET>
</BODY>
</HTML>
```

7 Bibliography

[1] Pressman, R. S., *Software Engineering – A Practitioner's Approach,* McGraw Hill, 2001; ISBN 0-07-365578-3.

[2] Beck, K., E*xtreme Programming Explained,* Addison-Wesley, 2000; ISBN 0-201-61641-6.

[3] http://Java.sun.com

[4] http://www.efunda.com

[5] http://flagpole.mit.edu