

An Object Oriented Approach to Matrix Analysis of Structures

By

Akshay R. Sthapit

BCE, Civil Engineering (2000)
Georgia Institute of Technology

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

at the

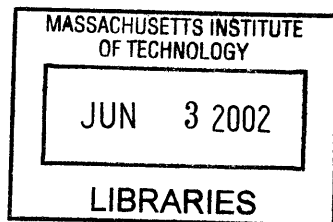
Massachusetts Institute of Technology
June 2002

© 2002 Massachusetts Institute of Technology
All rights reserved

Signature of Author
Department of Civil and Environmental Engineering
May 7, 2002

Certified by
Professor L. L. Bucciarelli
Professor of Engineering

Accepted by
Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies



BARKER

An Object Oriented Approach to Matrix Analysis of Structures

by

Akshay R. Sthapit

Submitted to the Department of Civil and Environmental Engineering
on May 7, 2002 in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Civil and Environmental Engineering

Abstract

The objectives of this study are twofold. The first is to develop programs that will allow users to easily create and analyze 3D Structures using the *Direct Stiffness Method*. Two programs have been developed - *Trussworks*, models truss structures, and *Frameworks*, models the more general framed structures that can carry and transfer shear forces and moments as well as axial loads. Also included is a database of steel cross-sections and their corresponding geometric properties developed by a colleague and adapted for the programs, as well as material properties of the most commonly used engineering materials.

The second objective is to explore the use of Object Oriented programming concepts in the structural analysis field. The language used in this study is Java, which is an elegant object oriented language introduced in 1993. The idea of objects is a powerful concept that, if properly used, makes for programs that are robust, stable, efficient, and easily extensible.

These programs are intended to be free to use, the primary target audience being undergraduate engineering students and faculty. The programs, and source code, as well as the documentation, can all be downloaded free of cost. The authors' intend this to be an "open source" product, available for modification and expansion to suit any user's needs. While there is no limit to the number of nodes and members that might be modeled, these are not intended to function as industrial strength programs for structural engineering professionals - but they may be of use to the latter in preliminary design. Frameworks and Trussworks can be run both as an application on a users local machine, or as an applet on any web-browser.

Thesis Supervisor: Dr. L. L. Bucciarelli
Title: Professor of Engineering

Table of Contents

<u>1</u>	<u>INTRODUCTION</u>	<u>7</u>
1.1	HISTORICAL BACKGROUND	7
1.2	APPROPRIATE COMPUTER USAGE FOR STRUCTURAL ANALYSIS	8
1.3	DESIGN ISSUES FOR EDUCATION	8
1.4	ABSTRACTION OF STRUCTURES AND LOADS	9
<u>2</u>	<u>OBJECT ORIENTED PROGRAM DESIGN</u>	<u>12</u>
2.1	OVERVIEW AND PHILOSOPHY	12
2.1.1	TRADITIONAL METHODS	12
2.1.2	REUSE	12
2.1.3	TOP-DOWN DESIGN	13
2.1.4	SYSTEM INTEGRITY	13
2.2	TECHNICAL PRINCIPLES OF OBJECTED ORIENTED PROGRAMS	14
2.2.1	ENCAPSULATION	14
2.2.2	INHERITANCE	16
2.2.3	POLYMORPHISM	19
2.3	USING OBJECTS FOR STRUCTURAL MODELING AND ANALYSIS	23
2.3.1	PACKAGE STRUCTURE	24
2.3.2	CLIENT SERVER MODEL	31
2.4	XML CONCEPTS	32
2.4.1	OVERVIEW OF XML TECHNOLOGY	32
2.4.2	USING STRUCTUREXML TO CREATE, STORE AND RETRIEVE STRUCTURES	33
2.5	OPEN SOURCE AND IT'S IMPLICATIONS	37
2.6	A DEVELOPER'S PERSPECTIVE	38
2.6.1	ADD MORE MATERIAL PROPERTIES	39
2.6.2	EDIT OR ADD CROSS-SECTIONS	39
2.6.3	VIEW AND EDIT THE STRUCTURE IN 3D IN A GRAPHICAL INTERFACE	39
2.6.4	USING A SPREADSHEET TO CREATE THE STRUCTURE	40
<u>3</u>	<u>STUDENT USE</u>	<u>41</u>
<u>4</u>	<u>CONCLUSIONS</u>	<u>51</u>
<u>5</u>	<u>REFERENCES</u>	<u>52</u>
<u>6</u>	<u>APPENDIX</u>	<u>53</u>

6.1	TUTORIAL AND DEMO OF A SAMPLE PROBLEM	53
6.1.1	PROBLEM STATEMENT	53
6.1.2	SET UNITS	53
6.1.3	CREATE THE STRUCTURE	54
6.1.4	ENTER MEMBER PROPERTIES	56
6.1.5	CONSTRAIN NODE	57
6.1.6	LOAD NODE	59
6.1.7	VIEW DISPLACED STRUCTURE	59
6.1.8	VIEW MEMBER FORCES	60
6.1.9	VIEW REACTIONS	61
6.1.10	TEXT OUTPUT	62
6.1.11	VIEW STIFFNESS MATRIX	63
6.1.12	SAVE THE STRUCTURE	64
6.2	FUNCTIONS AVAILABLE	65
6.2.1	FILE	66
6.2.2	GRAPHICS	68
6.2.3	FUNCTION	69
6.2.4	COMPUTE	76
6.3	THE STIFFNESS MATRIX	81
6.4	SYSTEM REQUIREMENTS TO RUN PROGRAM	84
6.4.1	AS AN APPLET	84
6.4.2	AS AN APPLICATION	84

1 Introduction

1.1 Historical Background

Structural Engineering has come a long way since the introduction of computers. Formerly, it required an army of engineers to perform the task of analyzing large systems (many of whom were simply checking each others results). This can now be done with more efficiency and accuracy with the use of appropriate computations.

Computers are especially well suited for tasks that can be broken down into a series of well-defined steps. Computers do not get tired, nor are they affected by emotions, or distracted by the environment (not for now, and not in computers used for structural analysis anyway). We are still some time away from the day when some claim computers will take over and analyze and design physical infrastructure without the aid of the engineer, but we are at the stage that once a structure has been correctly abstracted and modeled, the computer can process the data for hours, at the end of which it can faithfully and correctly predict how the loading conditions will produce displacements in the structure and how much force each member in the structure experiences. More “sophisticated” software, with building codes and member cross-sectional properties programmed into it, can even give reasonable suggestions for the type of members to use in the system.

James C. Maxwell who formulated the method of consistent deformations in 1864, and George A. Maney, who developed the slope-deflection method in 1915, laid the groundwork for matrix methods of structural analysis [Kassimali 99]. Before the use of computers, these methods had the disadvantage of requiring direct solutions of algebraic equations – which is a formidable task for a system with more than a few degrees of freedom and unknowns.

Once computers were developed, the task of solving large systems of simultaneous equations was suddenly eased, and people were quick to take advantage of this new capability. S. Levy is considered to be the first to introduce the flexibility method in 1947, which generalized the classical method of consistent deformations. In the early 1950’s, H. Falkenheimer, B. Langefors, and P.H. Denke extended the flexibility method to be conveniently expressed in matrix form. From that point, the field evolved rapidly and in the same year, J. H. Argyris and S. Kesley formulated matrix methods based on energy principles. In 1956, M.T. Turner, R. W. Clough, H.C. Martin, and L.J. Topp derived stiffness matrices for truss and frame members using the finite-element approach, and introduced the *direct stiffness method* for generating the structural stiffness matrix. This is the same method used in this study, and is still being used in most design offices today in the analysis of structures.

The early matrix methods were limited to analyzing framed structures only. But now finite element analysis procedures have evolved from these methods to overcome this restriction. Finite element analysis is an active field of research still today; structures of any form or shape can now be analyzed. The basic difference between finite element and matrix methods is that, in matrix methods, the relationships among force and displacement of an element are based on exact solutions of the underlying differential

equations, whereas, in finite element methods, such relations are generally derived by variational principles using assumed displacement or stress functions [Kassimali 99].

1.2 Appropriate Computer usage for Structural Analysis

The rewards of using computers for structural design are many. Today's buildings and bridges are larger and more intricate than ever before imagined possible in the pre-computer era. Still it is unwise to place all of one's trust in a piece of software. A program may produce artfully rendered models of buildings, color coded graphs, and pages upon pages of output, but the results of the analysis are only as good as the model. There may also be flaws in the software itself.

Public safety, health, and lives are at stake in designing office buildings, schools, bridges, and airplanes, not to mention the space station. It is the duty of the structural engineer to be competent in structural modeling, analysis and design, and not rely blindly on the results generated by a computer program. Structural engineers must understand the basic principles of matrix analysis in order to understand how commercially available software works, and if need be develop their own computer programs. It is in this spirit that the software developed in this project is made as open as possible so that should a user wish to do so, it is possible to read the source code, understand how it works, and perhaps make improvements as needed.

1.3 Design Issues for Education

Software designed for students must address different issues than those designed for professionals. The goal of educational software is to teach, to encourage the user to explore and learn, and perhaps be "fun". This means that the user interface design is critical since it cannot be assumed that the user will spend much time learning how the program works. It is even possible that a user would want to use the software only once in order to understand a concept or help to solve a homework problem. Therefore the time required of the student to learn to use the program should be as short as possible, or else he or she may lose interest.

Professional software, on the other hand, accommodates a different set of priorities. A professional grade structural analysis tool must be robust. The user interface is also important, but from a different perspective – most assume that the user has an incentive to spend time learning the software if it means efficiency later on. These programs usually have "room to grow" and come filled with shortcuts, and hidden toolbars and menu items, which can be turned on once the user is more familiar with the software.

This does not mean, however, that the educational programs have to be any less powerful. Trussworks, and Frameworks, for example, set no limit on the number of elements that can be added to a structure. And if users feel clicking and dragging a mouse to be inefficient for modeling structures with large numbers of members, then they can use the XML feature, which allows the use of a spreadsheet to enter data, which can then be exported in the StructureXML format.

1.4 Abstraction of Structures and Loads

Taking a structure that exists in the real world and reducing it for analysis by computer is a process by which the essential properties of the physical entity is idealized into a mathematical model that is amenable to analysis. To understand this process, one must consider developing the models in steps or *level of abstraction* [Kausel, Roesset 00].

A two-story frame has been used here as an example (Figure 1-1). The structure has been subjected to lateral wind loads and is rigidly connected to the ground. As it is, the structure contains infinite degrees of freedom, which after three levels of abstraction gets reduced to 18 degrees of freedom. Here degrees of freedom of a structure are defined as “*the independent joint displacements (translations and rotations) that are necessary to specify the deformed shape of the structure when subjected to an arbitrary loading*” [Kassimali 99].

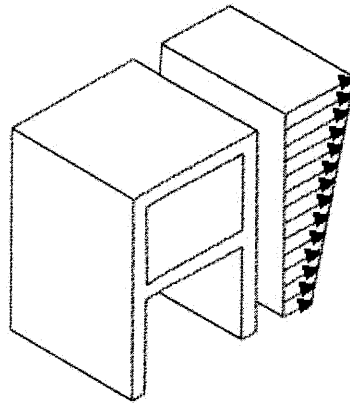


Figure 1-1 - Two Story Frame under wind loading

In the first level of abstraction, the wind loads are modeled as a linearly varying distributed load along one side of the building, and the building is modeled as a simple framed structure that is connected rigidly at all connections (Figure 1-1). This is a continuous system with has an infinite degrees of freedom.

The second level of abstraction converts the building into a system connected to one another through framed members that are also connected rigidly to one another (Figure 1-2). These framed members have associated with them the mass and stiffness of the original structure and considered “line” members that connect to one another at the nodes.

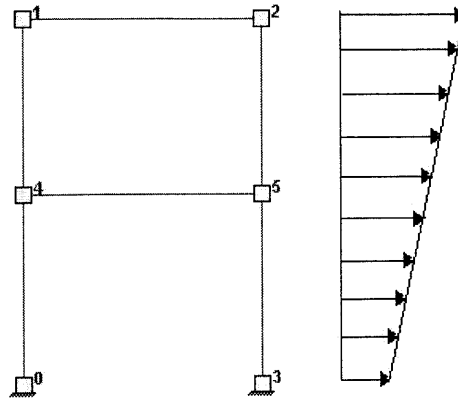


Figure 1-2 - Idealized system

The last step involves converting the distributed loads to nodal point loads. The wind loads are converted to nodal point loads using simple tributary area considerations (Figure 1-3). Our model is now a discrete one with 6 nodes, and therefore 6 times 3, or 18 degrees of freedom.

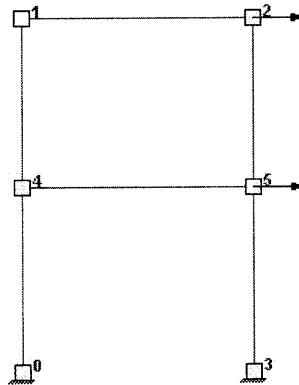


Figure 1-3 - Discretized System

Finally, the model is created on the program, Frameworks, and the deformed structure as computed by the program is plotted (Figure 1-4).

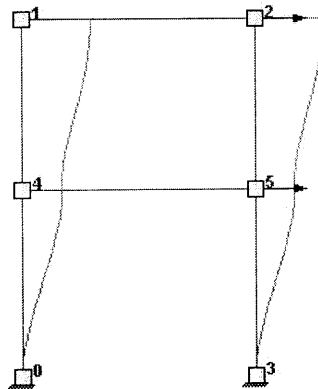


Figure 1-4 - Deformed Structure after Analysis

In our study, structures are grouped into two categories. The first is a space frame, which is the more general case. Members are connected to one another, and to the ground by rigid connections, which may or may not be constrained. Point loads and moments can be applied on any joint. These members can be subjected to bending moments, shears, torsion, as well as axial forces. Frameworks has been developed to handle this category of structures.

The other category is a space truss, which are idealized members connected to one another and to the ground through frictionless ball and socket joints. As a result, truss members can only carry forces along the member as tensile or compressive forces, and are incapable of carrying and transferring moments. A great many number of structures such as radio towers, bridges, and roof structures can be modeled as space trusses as they can be assumed to carry most of their loads as axial loads and the moments and shear forces can be safely neglected. Trussworks has been developed to model such truss structures.

2 Object Oriented Program Design

2.1 Overview and Philosophy

Most computer programs, including Trussworks, and Frameworks, digitally represent systems. Programs may be complex in nature, and in most instances the inner workings need not be understood thoroughly by the user to be able to properly use it. This is much like someone who can successfully drive a car without understanding all the details that go into making or maintaining the vehicle. The user is taught how to drive given certain interfaces, e.g., the steering wheel, the gas pedal, brakes, and the gear shifting mechanism. Most humans have a limited ability for dealing with more than one thing at a time, and we have to rely on shortcuts and heuristics to perform effectively. We need clean interfaces that connect us with much of the complex machinery that makes up life today, or else risk being overwhelmed with details.

2.1.1 Traditional methods

A computer programmer is like a car driver. Programs can grow so large in size that it becomes a burden to keep track of all the variables and methods involved. Not too long ago, however, this was the only method available to users of traditional programs such as those written in C and FORTRAN. These programs are *procedural* in nature and are structured such that they run linearly, from start to end. As the size of programs written in these languages grows, so does the complexity in keeping track of all the incorporated variables and methods. Such programs become unmanageable over time, errors compound, and extending or changing the code becomes a formidable task.

2.1.2 Reuse

The *Object oriented* approach seeks to break the programming task into manageable chunks. This way a project can be attacked from different angles, perhaps by a team of programmers. Once it is known what a certain part of the code does, one can concentrate on programming that module and not worry about complications with the rest of the system other than at the interfaces. Once that is done, each part, or “class”, will then do what it needs to do and the user can set that piece of code aside and work in the future only with the interface requirements.

Classes, however, are not merely subroutines in a program. They are a much more powerful category that allows code to be reused. A good analogy to a base class would be a nut, which once manufactured, can be used in making a house, a bridge, or a car. In the same way, once a class is constructed, other programs can use that class by knowing about its “public” interface, a feature known as *abstraction*. Abstraction allows the outside world to work with a class using its interface and not have to worry about the innards of the program.

To illustrate the power of this concept, consider a table, how it is made of four legs, and a top surface. Imagine how convenient it would be if you could just make *one* leg, and then make three more replicas of it, rather than crafting each leg individually. Not only that, imagine for a moment if every other table manufacturer in the world could use your leg class to make their tables and not worry about how to make a leg! Even better,

make a table template (class) and then no one would ever even have to worry about working at the leg level, and instead simply make instances of the higher table class. This is one of the objectives of the object-oriented approach – reuse good code so that you are not constantly reinventing the wheel and starting from scratch. Stand on the shoulder of giants one might say.

2.1.3 Top-down design

Object oriented programming allows users to work abstractly, which is a significant advantage in software engineering. Thinking at the abstract level helps the programmer to first fully understand the problem before any coding is done. Using this approach, the details and implementations are programmed at a later time, after initially working at a higher level to properly decide on the objects and their functionality.

This design method reduces the chances of code having to be re-written after it is found that it is not doing what it is supposed to do, or more commonly, that the program needs to address different issues from what the code is programmed to do. Code rewrite is one of the major problems in a software design project, which can cause significant delays, and lowering of team morale. The top-down approach has been used successfully to address these issues, and the object-oriented environment encourages this method.

2.1.4 System Integrity

It is a requirement for most software to be able to evolve to accommodate additional requirements at a later time. Updating a system does not have to be a complicated task if one adopts an object-oriented approach. Since we are dealing with independent objects that interact with other objects and with the user via known interface rules, one can safely decommission parts of the system, and replace them with new modules or classes when the need arises. If the new classes come with the same interfaces, it can be assured that the integrity of the whole system will not be compromised by the new code.

If you had a flat tire, or needed a new one, you would, without giving it a second thought, replace the old tire and drive with the knowledge that the car would still perform as well, if not better, than previously. It would be absurd to buy a whole new car simply because of a flat tire. You wouldn't worry about any tire-car incompatibility if you checked beforehand the required size and then bought that tire. Replacing old classes is the exact same process - you introduce new classes that have the same interface as the old ones.

Employing object-oriented methods also enforces good programming techniques, e.g. designing each module to do its job independently and avoiding global variables. In this way, modifying these modules at a later time will not create any unwanted side effects in the main program because will handle internal data “privately” or locally, and simply accept and return data using public interfaces without infringing on, or corrupting the data of other classes. In short, new versions of a program can be produced, safely, quickly, and without a lot of heartache if classes are used to divide and conquer the task at hand.

2.2 Technical Principles of Objected Oriented Programs

2.2.1 Encapsulation

At the fundamental level, all programs are made of data and instructions. In the traditional model, data is stored in memory and manipulated by code or subroutines. “Encapsulating” code that deals with the data and its storage is at the heart of the object-oriented design paradigm.

Simply put, encapsulation puts a protective front to data and code, and prevents it from being arbitrarily accessed from the outside. This makes the “private” data secure and ensures that it serves its purpose without interference from outside. The outside world can then work with the data using the classes’ public interface ensuring that the data is being handled the way it should be. For example, you can change gears in your car using the car’s public interface – the gear stick and the clutch. This ensures that the car will behave the way you want it to, or more rather, the way it is supposed to. It would not do much good, and indeed would be unsafe, if it were possible to shift gears using the turn signals.

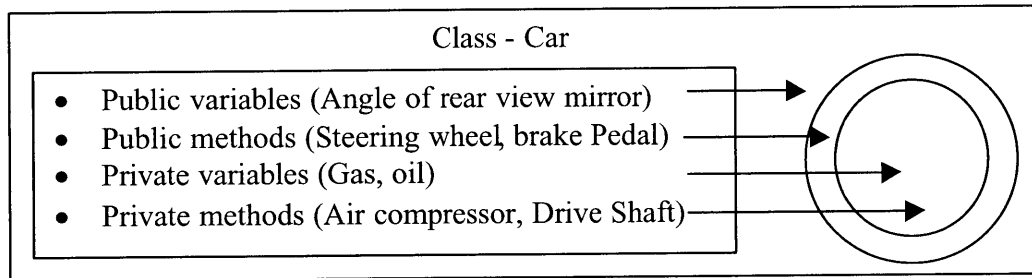


Figure 2-1 – Encapsulation used to protect private data and methods

In Java, encapsulation is made possible through the use of classes. Classes are simply templates that define methods and data. Instances of a class create objects, which interact safely with the outside world through the use of *public* methods. *Private* methods and data, in contrast, as the name suggests, are private, and cannot be corrupted from outside the class. Java provides two more access levels – *protected* and *package*. Only subclasses and the class itself can access *protected* variables and methods. The *package* access level allows only classes within the package (Section 2.3.1) to access the data and methods. The various access levels are summarized in Table 2-1.

Specifier	Class	Subclass	Package	World
Private	●			
Protected	●	●	●	
Public	●	●	●	●
Package	●		●	

Table 2-1 - Access Level Summary

As an example, Trussworks and Frameworks both use the Member.java class. This class is comprised of the variables and methods as shown in Table 2-2 and Table 2-3. For example, since memberID is a private integer, it cannot be accessed or changed from

outside the `Member.java` class arbitrarily. The only way to do that would be to use its public methods `getMemberID()` and `setMemberID()`. These methods that get and set private variables are frequently called *accessor* and *mutator* methods.

Summary of Variables of Member.java		
Access Level and Type	Variable Name	Description
Private int	<code>memberID</code>	Keeps track of the member ID
protected Joint	<code>jStart, jEnd</code>	Each member comprises of two joints (nodes)
private double	<code>L</code>	Private variable records length of member
protected double	<code>sFx, efx</code>	Private double stores axial forces in member
private Boolean	<code>sMR, eMR</code>	Private boolean keeps track of moment release in member
protected static int int numberOfMembers	<code>numberOfMembers</code>	Counter that stores the number of members in the structure

Table 2-2 – Summary of Variables of Member.java

Method Summary of Member.java	
Return Type	Method Name
Double	<code><u>getEFx()</u></code>
Boolean	<code><u>getEMR()</u></code>
<u>Joint</u>	<code><u>getJEnd()</u></code>
<u>Joint</u>	<code><u>getJStart()</u></code>
Double	<code><u>getL()</u></code>
Int	<code><u>getMemberID()</u></code>
static int	<code><u>getNumberOfMembers()</u></code>
Double	<code><u>getSFx()</u></code>

boolean	<u>getSMR</u> ()
static void	<u>reset</u> ()
Void	<u>setEFx</u> (double eFx)
Void	<u>setEMR</u> (boolean eMR)
Void	<u>setJEnd</u> (<u>Joint</u> jEnd)
Void	<u>setJStart</u> (<u>Joint</u> jStart)
Void	<u>setMemberID</u> (int memberID)
Void	<u>setSfx</u> (double sFx)
Void	<u>setSMR</u> (boolean sMR)
java.lang.String	<u>toString</u> ()

Table 2-3 – Summary of Public Methods in Member.java

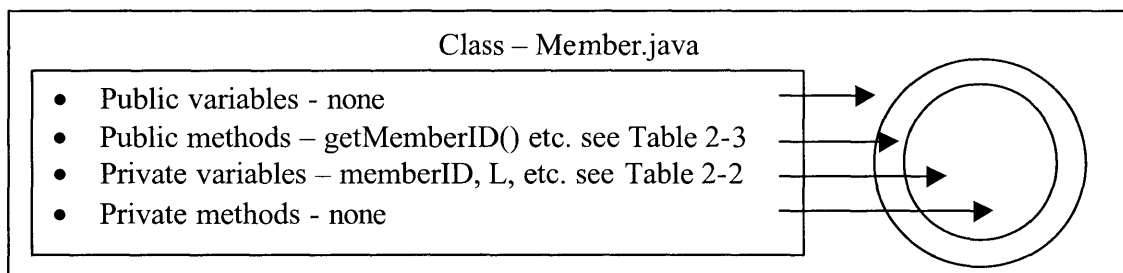


Figure 2-2 – Encapsulation used to protect private data and methods in Member.java

2.2.2 Inheritance

Most things can be viewed hierarchically. For example a Labrador dog is a *class* of Canines, who are Mammals, which are of type Animal (Figure 2-3). This forms a *class hierarchy* that allows us to properly categorize a certain class relative to its subclass and superclass. In this example, Canines are a subclass of Mammals, and a superclass of Labradors.

Inheritance allows the subclasses to provide specialized behaviors using the common elements from the superclass. The code in the superclass can then be reused many times by the classes inheriting the class. A superclass can also be an *abstract* class, which define “generic” behaviors but leaves it to the user to implement them. These abstract

classes provide the class concepts, but do not come with the code that implements them. It is left to the user inheriting, or “extending” these classes to provide the code that defines the abstract classes and methods [Campione 00]. Abstract classes both let users work at a higher level, as well as ensure that subclasses will behave in certain way.

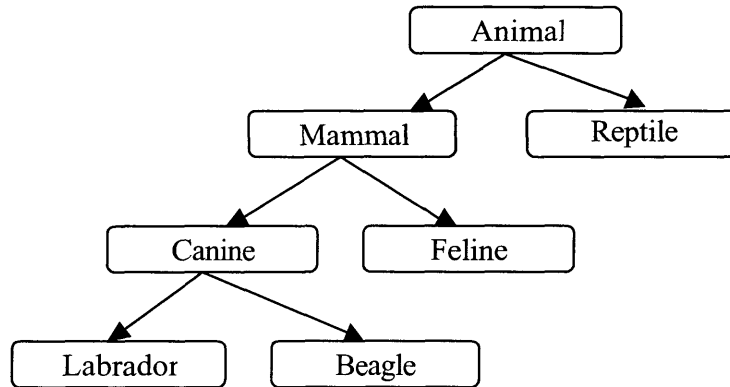


Figure 2-3 – Class Hierarchy and Inheritance

Inheritance works hand-in-hand with the concept of encapsulation. Subclasses inherit the encapsulation properties of the superclass, and can include specialized encapsulated methods and data themselves. This has the benefit that it makes known in advance how a subclass will behave. A Labrador will always have the characteristics of a mammal, and in the same way, programmers can use subclasses with the knowledge that they inherit the base classes’ methods and data.

Frameworks and Trussworks both utilize this concept of inheritance. In fact, since Java is a purely object oriented language, it would be impossible not to. Everything in Java is treated as objects, and all classes extend the `java.lang.Object` class by default. As an example, the `Member.java` class extends the `java.lang.Object` class, which is in turn extended by `Frame.java` (Figure 2-4).

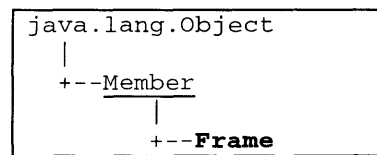


Figure 2-4 - Class Hierarchy for Frame.java

`SimPad.java`, is another example (Figure 2-5) which inherits `java.awt.Frame`. `SimPad` is used in Trussworks and Frameworks to display text, such as for showing the stiffness matrix and text output of the results (Section 2.3.1). It can be seen that `java.awt.Frame` itself inherits many subclasses. The advantage then is that `SimPad.java` has access to all the public and protected methods and variables of the superclasses without the need of any additional programming.

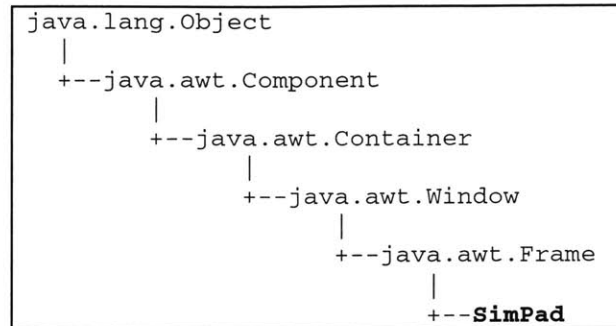


Figure 2-5 - Class Hierarchy for SimPad.java

Subclasses can also “implement” abstract classes as mentioned above. DrawPanel.java implements the following “interfaces” (Figure 2-6). For example, since DrawPanel.java implements the java.awt.event.MouseListener interface, it has to define what happens when *all* of the MouseListener methods are called (Table 2-4). And since it may sometimes be tedious to implement all the methods of the interface, Java also provides “Adapters”, which simply implement interfaces and define its methods as empty methods that do nothing. Thus a user can create an instance of this adapter class and simply over-ride the methods (see Section 2.2.3.2) that are needed. MainFrame.java, for instance, extends java.awt.Frame, and we need some way of making the Frame or Window closable. Rather than implement the WindowListener interface, which would require defining all of its methods, we simply create an instance of the WindowAdapter.java class, and then over-ride the windowClosing() Method (see Figure 2-7).

```

javax.accessibility.Accessible, java.util.EventListener, java.awt.image.ImageObserver,
java.awt.event.KeyListener, java.awt.MenuContainer, java.awt.event.MouseListener,
java.awt.event.MouseMotionListener, java.io.Serializable
  
```

Figure 2-6 – Interfaces implemented by DrawPanel.java

Method Summary of java.awt.event.MouseListener	
Return Type	Method Name
void	<u>mouseClicked</u> (MouseEvent e) Invoked when the mouse has been clicked on a component.
void	<u>mouseEntered</u> (MouseEvent e) Invoked when the mouse enters a component.
void	<u>mouseExited</u> (MouseEvent e) Invoked when the mouse exits a component.
void	<u>mousePressed</u> (MouseEvent e) Invoked when a mouse button has been pressed on a component.

void	<u>mouseReleased</u> (MouseEvent e) Invoked when a mouse button has been released on a component.
------	---

Table 2-4 – Method summary for java.awt.event.MouseListener

```

public MainFrame(Button startButton) {
    setTitle("TrussWorks - untitled.truss");

    addMenu();
    addPanels();

    addWindowListener(new WindowAdapter() { //make window
                                                //closable
        public void windowClosing(WindowEvent e) {
            dispose();
            System.exit(0);
        }
    });

    drawPanel.reset();

    this.startButton = startButton;

    setSize(700,500);
    setVisible(true);

    drawPanel.setInitialValues();
}

```

Figure 2-7 – An example of an adapter in MainFrame.java

2.2.3 Polymorphism

As the name suggests, polymorphism means that classes in an object-oriented environment can have different shapes. It is a useful feature that allows a piece of code to behave differently based on the task at hand. There are two ways this can be done.

2.2.3.1 Overloading

This means that based on the type, or number of parameters that is passed to a method, a piece of code will behave differently, or return a different type of variable. Two methods may have the same name, but the program is smart enough to know what needs to be done when dealing with different types of data. For example, you can program a function `getArea(Object o)`, which return the area πr^2 when dealing with a circle object with radius r , or which returns bh when dealing with a rectangle object with a length of b , and height h .

`TrussWorks.Tools.Compute.java`, for example, overloads the `closeEnough()` method such that it computes the distance between a point and another point, or a member, or a node respectively (Figure 2-8) and returns true if it is less than a certain distance. The compiler is smart enough to know which method to use based on the argument supplied.

```

public static boolean closeEnough(double x,double y,double x0,double
y0) {
    if (Math.sqrt((x0-x)*(x0-x)+(y0-y)*(y0-y)) <= radius) return
true;
    else return false;
}

public static boolean closeEnough(double x, double y, Joint j) {
    double dist;
    double x0 = j.getX();
    double y0 = j.getY();
    if (Math.sqrt((x0-x)*(x0-x)+(y0-y)*(y0-y)) <= radius) return
true;
    else return false;
}

public static boolean closeEnough(double x3, double y3, Member member)
{
    double x = Compute.closestX(x3,y3,member);
    double y = Compute.closestY(x3,y3,member);

    if (Compute.closeEnough(x3,y3,x,y)) return true;
    else return false;
}

```

Figure 2-8 – `closeEnough()` method overloaded in `TrussWorks.Tools.Compute.java`

Constructors may also be overloaded in the object-oriented environment. This allows “different” objects to be instantiated based on the parameters passed to the constructor. `Joint.java`, for example, has five constructors (Figure 2-9). The third constructor, `public Joint(int JointID, double x, double y)`, creates a two dimensional `Joint` object, while the fourth constructor, `public Joint(int JointID, double x, double y, double z)`, creates a three dimensional `Joint` object.

```

public Joint() {
    this.x = 0;
    this.y = 0;
    this.z = 0;
    this.Fx = 0;
    this.Fy = 0;
    this.Fz = 0;
    this.FMx = 0;
    this.FMy = 0;
    this.FMz = 0;
    this.Rx = false;
    this.Ry = false;
    this.Rz = false;
    this.Mx = false;
    this.My = false;
    this.Mz = false;
    NumberOfJoints++;
}

public Joint(int JointID) {

```

```

        this();
        this.JointID = JointID;
    }
    public Joint(int JointID, double x, double y) {
        this(JointID);
        this.x = x;
        this.y = y;
    }
    public Joint(int JointID, double x, double y, double z) {
        this(JointID,x,y);
        this.z = z;
    }
    public Joint(Joint joint) {
        this(joint.JointID, joint.x, joint.y, joint.z);
    }

```

Figure 2-9 – Constructor overloading in Joint.java

2.2.3.2 Overriding

When inheriting classes, it may not always be desirable for the subclass to implement the same methods. Rather than forcing the programmer to reprogram the entire class, overriding methods allows a user to make the inherited method in the subclass do something else. For example, a class called point2D may have a method called distanceToOrigin() which return the distance from the two dimensional point to the origin as $\sqrt{x^2 + y^2}$. If someone extends point2D and makes a point3D class, it would be easy to override the distanceToOrigin() method so that it returns the correct the correct distance to the origin, $\sqrt{x^2 + y^2 + z^2}$, instead.

One method which is commonly over-ridden in Java programs handling graphics is the paint() method in Panel.java. Doing so lets the user draw on the panel using the java.awt.Graphics class. Figure 2-10 shows how Drawpanel.java over-rides the paint() method which it inherited from the java.awt.Panel.java class to draw the structure on the screen.

```

public void paint(Graphics g) {
    .
    .
    .

```

```
Vector uniqueJoints =
    Compute.createUniqueJointsVector(frameVector);

if (gridB) {
    Draw.grid(g,getSize().width, getSize().height);
}
if (scaleB) {
    Draw.scale(g,getSize().width, getSize().height);
}
if (zeroAzeroEB) {
    Draw.zeroAreaZeroEMembers(g, frameVector);
}
if (jointIDB) {
    Draw.jointID(g,uniqueJoints);
}

.
.
.
}
```

Figure 2-10 – Over-riding of the paint() method in DrawPanel.java

2.3 Using Objects for Structural Modeling and Analysis

A person driving a car is probably not thinking at the sub-atomic scale. He interfaces with what is needed such as the door handle, the steering wheel, the gear shift and so on. It would be an inefficient use of time and effort to be thinking about each atom that goes into every part, every nut that hold the parts together, and how all the parts work with each other to make the car run. The Object Oriented paradigm allows programmers in the same way to utilize objects to think and design from a higher level. A structure no longer has to be thought of as made up of an array of numbers and letters associated with member numbers, member cross-sectional information, node numbers, and node characteristics. Instead, it can be neatly broken into a hierarchy as shown in Figure 2-11.

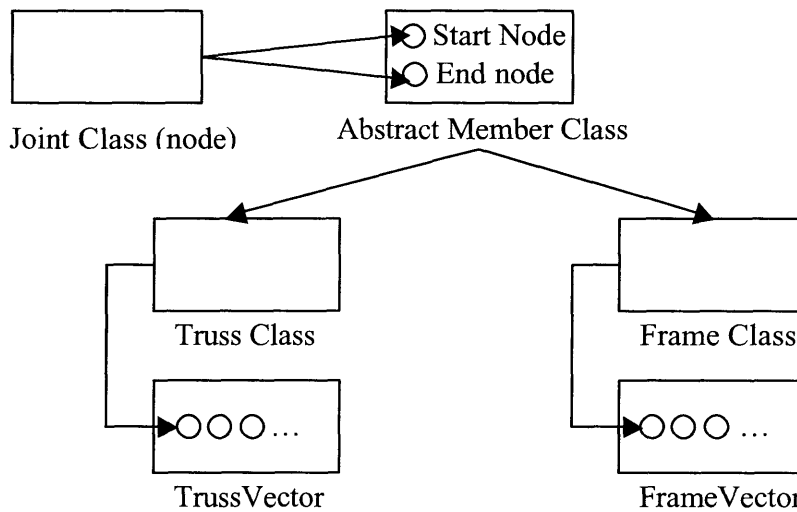


Figure 2-11 – Make up of a Structure in Trussworks and Frameworks

An analogy with a vehicle object might look something like shown in Table 2-5.

Joint Class	Nut
Abstract Member Class	Wheel spoke
Truss Class	Bicycle Wheel
Frame Class	Car Wheel
TrussVector	Bicycle
FrameVector	Car

Table 2-5 - Structure - Vehicle object analogy

The model shown in Figure 2-11 is the model utilized in Trussworks and Frameworks - each structure is made of members who have their own personal variables and methods associated with them. Every member, in turn is made of two nodes, which have their own set of variables and methods. But then we have two subsets - a Truss member extends the Member class to carry axial loads, while Frame members extend the

same Member class to carry shear and moments in addition to axial loads. Finally, these Truss and Frame objects are put in a TrussVector and FrameVector class, which simply store all the objects into a Vector.¹ These classes are then conveniently put in a package called “Support” (Section 2.3.1) that can be imported and used by other classes.

Other packages are created in the same way to perform specific tasks. The “Graphics” package handles all the graphics, the “Jama” (**J**ava **M**atrices) package does all the matrix computations, the “Sections” package has all the cross-sectional and material properties, the “Tools” package provides other miscellaneous tools required, and the “xml” package takes care of the XML parsing, input, and output. In the same way that a company has different departments, such as the human resources department, upper management, and sales that work together and function as whole, these packages perform specific tasks and coordinate together to function as a whole.

2.3.1 Package Structure

In Java, packages are a collection of classes. This helps make classes easier to find since they are associated with packages. Using a package also prevents name conflicts with classes written by other programmers, and helps control access to classes. If someone else wrote a `Joint.java` class, for instance, they could put in their package and use it in conjunction with Frameworks and Trussworks and not worry about their `Joint.java` class colliding with the `Joint.java` class included in Trussworks and Frameworks.

An example in Frameworks is the `Frame.java` class that defines a frame structure. `Frame.java`, however, is also defined in the core `Java.awt` package and is a window with a menubar and so on. So how does the compiler know which `Frame` you are referring to when you want to create a `Frame` object? Since they are both defined in separate packages, they can be referred to without any name conflicts by referring to them as `java.awt.Frame.java`, and `FrameWorks.Support.Frame.java`.

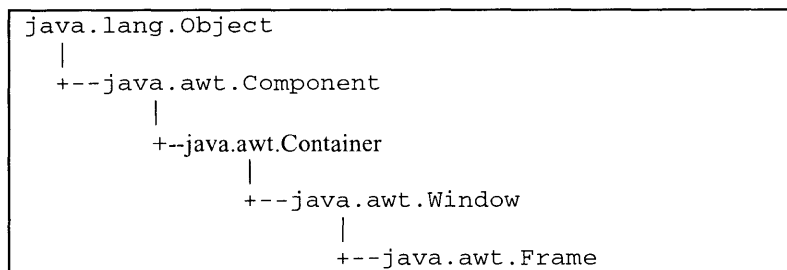


Figure 2-12 - `Frame.java` as defined by the `Java.awt` package



¹ A “vector” in Java behaves much the same way as an array, the difference being that Vectors can grow in size as needed. This reduces reallocation and wasted space. It is useful for programs like Trussworks since it is not known in the beginning how large to create the array to store the members. A user may create a 2 member structure, or a 2000 member structure. Using the Vector class, instead of an array, therefore places no limit on the number of member that may be added to the structure, while at the same time minimizing memory allocation. The program allocates only as much memory as is required.

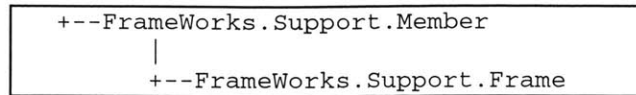


Figure 2-13 - Frame.java as defined by FrameWorks.Support package

In addition to the advantages mentioned above, packages also make it easier for other programmers to use your code. Instead of copying individual classes, they can simply *import* classes that they need. Figure 2-14, for example, shows that Frameworks package is made of six internal packages. These packages prevent any name conflicts, and also helps organize the classes neatly, which is much like using a filing system with folders to organize paperwork in an office. In all these instances, of course, we only deal with the public and protected interfaces or methods of each class in the set of packages.

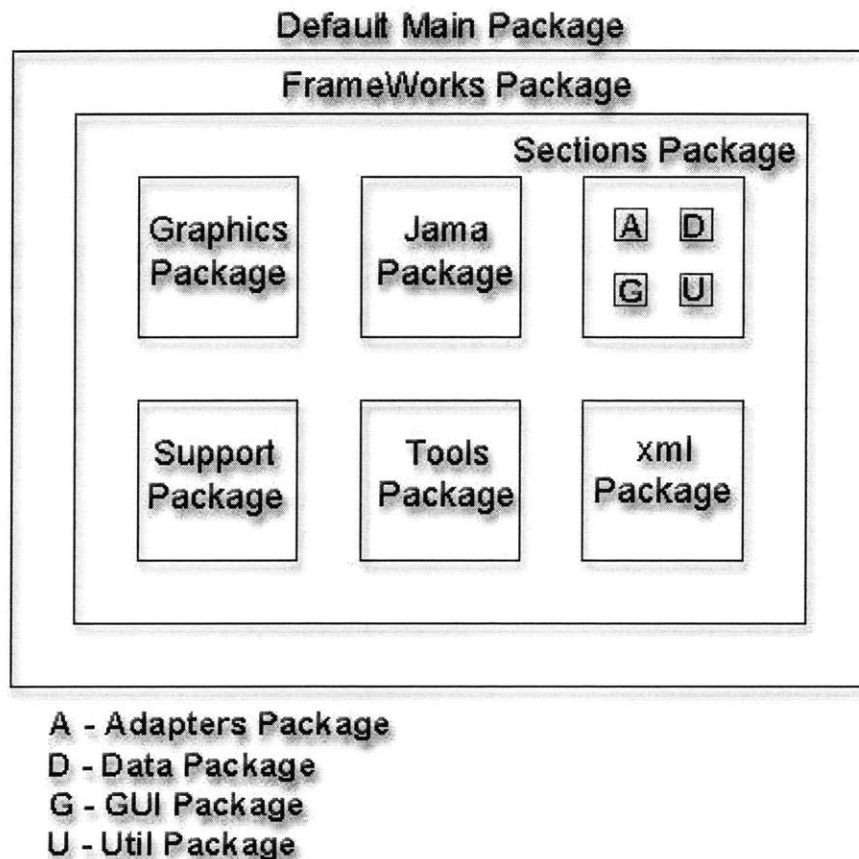


Figure 2-14 - Package Structure for Frameworks

2.3.1.1 Graphics Package

This package deals with the “graphics user interface” or the gui in the programs (See Section 6.1 for screen shots). It contains the following classes –

- `ConstrainDialog.java` – Displays the window that let users constrain nodes.

- `DialogBox.java` – Dialog box that deals with error messages, download and upload files, set initial screen dimensions and so on. The constructor is overloaded to display the correct dialog box.
- `DrawPanel.java` – The main window where the structure is drawn. This also handles all the mouse and keyboard input, including the panel on the bottom that displays information.
- `LoadDialog.java` – Displays the window that let users load nodes.
- `MainFrame.java` – The parent window that contains the `DrawPanel` object, and also acts as the main controller between all objects. This window contains all the menu-items.
- `MemberForcesDialog.java` – Displays the window that let users view member forces.
- `MemberReleaseDialog.java` – Dialog-box that let users release member moments.
- `XMLPad.java` – Window that displays `StructureXML`, and also parses the XML text using the `xml` package (Section 2.3.1.6).

2.3.1.2 Jama Package

This package deals with all the matrix computations in the programs. This software is a cooperative product of The MathWorks and the National Institute of Standards and Technology (NIST) which has been released to the public domain². A summary of the capabilities of JAMA is listed in Table 2-6.

Object Manipulation	constructors set elements get elements copy clone
Elementary Operations	addition subtraction multiplication scalar multiplication element-wise multiplication element-wise division unary minus transpose norm
Decompositions	Cholesky LU QR SVD symmetric eigenvalue nonsymmetric eigenvalue

² Further information about the package can be found at <http://math.nist.gov/javanumerics/jama>

Equation Solution	nonsingular systems least squares
Derived Quantities	condition number determinant rank inverse pseudoinverse

Table 2-6 - Summary of Jama Capabilities

2.3.1.3 Sections Package

This package deals with the steel cross-sectional data that was developed by a colleague and adapted for use in the program. It is further divided into four sub-packages – adapters, data, GUI, and util.

- Adapters – This package contains all the adapters used the Sections package, and extend the functionality of classes such as buttons.
- Data – This package contains all the steel cross-sectional data in text format. It also contains data about material properties.
- GUI – Deals with all the graphics used in the member properties panel, as well as the panel used for searching for a cross-section.
- Util – Contains classes for tasks such as string parsing and converting numbers to exponential format.

2.3.1.4 Support Package

This package contains the core classes that define the structure as defined in section 0. They all extend the “`java.io.Serializable`” interface so that they can be stored persistently as objects, either on the users hard drive, or on the server over the Internet. They may be “deserialized” later to resume working on a structure (Section 2.3.1).

- `Joint.java` – The Joint class defines the joints or nodes in the structure. It keeps track of the joint ID, its coordinates, and also keeps track of whether it is constrained or not, and stores the nodal forces applied to it.

<code>private int JointID</code>	Holds the joint (node) ID
<code>private double x, y, z</code>	Joint coordinates
<code>private double Fx, Fy, Fz</code>	Forces applied at the node
<code>private double FMx, FMy, FMz</code>	Moments applied at the node
<code>private double dx, dy, dz</code>	Nodal Displacements
<code>private double dmx, dmy, dmz</code>	Nodal Rotations (radians)
<code>private boolean Rx, Ry, Rz, Mx, My, Mz</code>	Nodal Constraints (true = constrained)
<code>private double rcx, rcy, rcz, rcmx, rcmy, rcnz</code>	Reaction forces

Table 2-7- Instance variables of Joint.java

- `Member.java` – This is the abstract Member class that has two *private* Joint objects, and keeps track of the member ID, member releases, and stores the axial member forces that are computed due to the applied nodal loads.

<code>private int memberID</code>	Member ID
<code>protected Joint jStart, jEnd</code>	Start and end nodes of the member
<code>private double L</code>	Length of the member (computed by the object from the start and end nodal coordinates)
<code>protected double sFx, eFx</code>	Start and end member releases (true = released)

Table 2-8 - Instance variables of Member.java

- `Frame.java` – This class extends the abstract Member class and has access to all the protected and public methods of the super Member class. This class stores the other member variables as shown in the table below.

<code>protected static int numberOfFrames</code>	Static variable keeps track of number of members in the structure
<code>private double A</code>	Member cross-sectional area
<code>private double E</code>	Member modulus of elasticity
<code>private double G</code>	Member shear modulus
<code>private double J</code>	Member Polar moment of inertia, or the Saint-Venant's torsion constant
<code>private double Iy</code>	Moment of inertia of member cross-section about local y axis
<code>private double Iz</code>	Moment of inertia of member cross-section about local z axis
<code>private double B</code>	Beta Angle or angle of roll (in degrees). Measured anti-clockwise looking down on member, from start joint to end joint angle between local y axis and global y axis
<code>private double sFy, eFy;</code>	Start and end shear force in y direction
<code>private double sFz, eFz;</code>	Start and end shear force in z direction
<code>private double sMx, eMx;</code>	Start and end moment in x direction
<code>private double sMy, eMy;</code>	Start and end moment in y direction
<code>private double sMz, eMz;</code>	Start and end moment in z direction

Table 2-9 - Instance variables of Frame.java

- `FrameVector.java` – This class holds all the members of the structure in a Vector, and keeps track of the file name.

<code>private Vector frameV</code>	Vector that holds member objects
<code>private String filename</code>	File name

Table 2-10 - Instance variables of `FrameVector.java`

2.3.1.5 Tools Package

This package does all the miscellaneous tasks such as computations, drawing the structure on the screen, and so on.

- `Compute.java` – Does all the computations for the program. For example, it computes the stiffness matrices, displacement vectors, and so on. It also helps with all the graphics calculations such as snap-to-grid coordinates.
- `Connection.java` – Deals with the connection to save files over the Internet. Determines the host name of the applet and so on.
- `CT.java` – Does all the coordinate transformations for the graphics by keeping track of the panning, and zooming.
- `df.java` – Class to format numbers into the exponential format.
- `Draw.java` – Draws the structure on the main panel, including loads, supports, and displaced structure.
- `Formatter.java` – Another class to convert numbers into the exponential format.
- `QuickSort.java` – Class to sort nodes during calculations.
- `SimPad.java` – Text area to display output, such as the stiffness matrices, text output, and error messages.

2.3.1.6 xml Package

This package handles all the XML (Section 2.4.1) parsing. It uses the `Ælfred` XML parser³ to parse XML in the `StructureXML` format. “`Ælfred` is a small, fast, DTD-aware Java-based XML parser, especially suitable for use in Java applets. [It has been] designed `Ælfred` for Java programmers who want to add XML support to their applets and applications without doubling their size: `Ælfred` consists of only two core class files, with a total size of about 26K, and requires very little memory to run.” It contains the following classes –

- `HandlerBase.java` - Convenience base class for `Ælfred` handlers.
- `SAXDriver.java` - This driver acts as a front-end for `Ælfred`, and translates `Ælfred`'s events into SAX events.
- `XmlException.java` - Convenience exception class for reporting XML parsing errors.
- `XmlHandler.java` - XML Processing Interface.

³ More information regarding the package can be found from the opentext website at <http://www.opentext.com>

- `XmlParser.java` - Parse XML documents and return parse events through call-backs.

2.3.2 Client Server Model

Figure 2-15 shows the model used in Trussworks and Frameworks to save and retrieve structures over the Internet, as well on the users local hard drive. All components of the structure such as the `Joint.java`, `Member.java`, and `Frame.java`, extend the `serializable` interface. This allows you to save the objects in its current state, which can then be retrieved at a later time. In our case, we save the structures either on the users local machine (when running as an application), or over the Internet using sockets. The later requires a dedicated server running the `Server.java` class, which is included in the package.

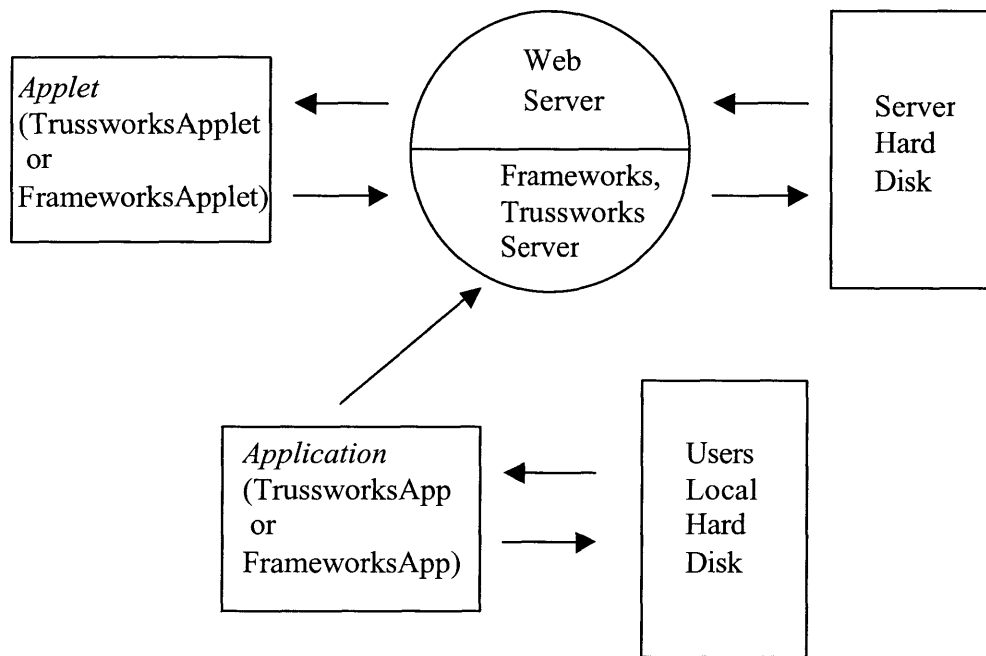


Figure 2-15 - Client Server Model to save and retrieve structures

Java restricts the connections that an applet can make and can only connect back to the same web server from where it originated. For this reason, the `Server.java` class must be run on the same web server from which the applet originates.

When running the programs as applications, however, connections can be made to any server with a valid host name.

2.4 XML Concepts

2.4.1 Overview of XML technology

Extensible Markup Language, or XML, is an attractive language for storing and delivering information, and it is especially well suited for use on the Internet. The World Wide Web Consortium (W3C) describes the language as follows: *“The Extensible Markup Language (XML) is a subset of SGML... Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.”* Many supporting technologies have been developed that work alongside with XML, such as XLink, DTD, CSS and so on [Connolly 01].

“XML is a method for putting structured data in a text file”. When we say “structured data”, we mean things such as spreadsheets, address books, technical drawings, or in our case, data that describes our structure such as node coordinates, member properties and so on. Trussworks and Frameworks both allow the user to save their structure in a *binary* format on a disk, either over the web (using the Client-Server model), or on the local hard drive when running the programs as an application. Saving the structure in a readable *text* format, however, allows you, to look at the data without the program that created it, if necessary. XML can be considered as a set of rules or guidelines, to “design the text format in such a way that it produces files that are easy to generate and read, that are unambiguous, and that avoid common pitfalls, such as lack of extensibility, lack of support for internationalization/localization, and platform-dependency” [Connolly 01].

Since the data file generated is a XML validated text file, another advantage is that it would be possible for a user to create third-party software that could interface with Trussworks and Frameworks. This would make it possible, for example, to enter data in a spreadsheet, and then let Trussworks and Frameworks do the calculations. Or, perhaps the XML structural data could be entered into a viewer program that would let people view the structure that they created within Frameworks and Trussworks in 3D and maybe zoom in, rotate, and pan the structure so that users can see the deformations as they would in “real life” in three dimensions.

XML makes use of tags (words bracketed by ‘<’ and ‘>’) and attributes (of the form name = “value”) and to the casual observer it may look like HTML. The difference is that HTML specifies what each tag and attribute means, which is often simply how the text will be displayed on a web browser. XML on the other hand, uses the tags only to define pieces of data. It is left to the application reading the data to interpret it. It is a bit like creating your own markup language, where the programmer decides what each tag and attribute stands for. For example, the XML structure used in Trussworks and Frameworks is StructureXML, where the <units> tag and the “length” and “force” attributes (Figure 2-16) are used to define the units used in the structure.

```
<StructureXML>
  <units length = "meters" force = "newtons"></units>
  .
  .
  .
</StructureXML>
```


Figure 2-16 - Sample StructureXML

Although XML files are text files, they are not designed to be read. The text files should ideally be looked as a way that lets “experts” debug applications in emergencies so that they can fix a broken XML file using a simple text editor. Another advantage to having text files is that they can pass safely through firewalls since it is hard to embed malicious code or viruses into a text file.

XML 1.0 is the specification that defines what “tags” and “attributes” are. Around XML 1.0, there are additional modules that provide tags and attributes for specific tasks. *Xlink*, for example, describes a standard way to add hyperlinks to an XML file. The Document Object Model, or *DOM*, “is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page” [Connolly 01]. There are other modules and tools available such as *XML Namespaces*, *XML Schemas* and so on, and more under development.

XML files are almost always larger than comparable binary formats since XML are text files that uses tags to delimit the data. The advantages of having the file as a readable text format as mentioned above, however, compensate for the extra storage, especially since disk space is not as expensive as it used to be, and it is also possible to compress files very efficiently using programs such as zip and gzip.

Another reason to use XML is that there are already a large and a growing community of tools available that support XML. This is an obvious advantage to using some proprietary technology since you have the freedom to build your own software around it and being license free, to do so at no cost.

2.4.2 Using StructureXML to create, store and retrieve structures

In a few instances, it may be more convenient to create the structure using a text input file, or in the form of a spreadsheet. For example, if the number of members is large, the click and click interface may prove to be cumbersome. Or in the case of a three dimensional frame or truss, a text based input may be the only choice.⁴

StructureXML not only fills this need, but it can also serve as the middle ground of communication between Frameworks and Trussworks, and another program. One possibility is that could create a graphics package that displays the structure by interfacing with the StructureXML text data.

XML provides another method of storing the structure in addition to the binary format discussed in section 2.3.2. Since these are simply text files, they carry further advantages. For instance, these text files can be very easily copied and pasted while running the programs as applets, or the text can be emailed without the need for attachments.

⁴ The program in its current state does all the calculations in three dimensions.

StructureXML was created such that the text file needs to be both well-formed, and valid.

- A well-formed document is one that meets the minimal set of criteria for a conforming XML document. For example, the start tag must be ended by an end tag. It must also contain the prolog and the document element (also known as the root element).
- A valid XML document has two additional requirements to that of a well-formed document. First, the prolog of the document must include a proper *document type declaration*, which contains a *document type definition* (DTD). Second, the rest of the document must conform to the structure defined in the DTD.

The DTD for StructureXML is defined in Figure 2-17. All StructureXML documents must have this at the beginning of the text file.

```

<?xml version="1.0"?>
<!DOCTYPE StructureXML
[
  <!ELEMENT StructureXML (units, uniqueJoints, members)>
  <!ELEMENT units EMPTY>
  <!ATTLIST units length (meters | centimeters | feet | inches)
#REQUIRED
                                force (newtons | kiloNewtons | pounds | kips)
#REQUIRED>
  <!ELEMENT uniqueJoints (Joint*)>
  <!ELEMENT Joint (coordinates, constraints, loads)>
  <!ATTLIST Joint JointID ID #REQUIRED>
  <!ELEMENT coordinates EMPTY>
  <!ATTLIST coordinates x CDATA #REQUIRED
                                y CDATA #REQUIRED
                                z CDATA #REQUIRED>
  <!ELEMENT constraints EMPTY>
  <!ATTLIST constraints Rx (true | false) #REQUIRED
                                Ry (true | false) #REQUIRED
                                Rz (true | false) #REQUIRED
                                Mx (true | false) #REQUIRED
                                My (true | false) #REQUIRED
                                Mz (true | false) #REQUIRED>
  <!ELEMENT loads EMPTY>
  <!ATTLIST loads Fx CDATA #REQUIRED
                                Fy CDATA #REQUIRED
                                Fz CDATA #REQUIRED
                                FMx CDATA #REQUIRED
                                FMy CDATA #REQUIRED
                                FMz CDATA #REQUIRED>
  <!ELEMENT members (Member*)>
  <!ELEMENT Member (properties, memberRelease)>
  <!ATTLIST Member jStart CDATA #REQUIRED jEnd CDATA #REQUIRED>
  <!ELEMENT properties EMPTY>
  <!ATTLIST properties A CDATA #REQUIRED
                                J CDATA #REQUIRED

```

```

Iy CDATA #REQUIRED
Iz CDATA #REQUIRED
E CDATA #REQUIRED
G CDATA #REQUIRED
B CDATA #REQUIRED>
<!ELEMENT memberRelease EMPTY>
<!ATTLIST memberRelease sMR (true | false) #REQUIRED
                        eMR (true | false) #REQUIRED>
]
>

```

Figure 2-17 - DTD for StructureXML

The body of a sample StructureXML document is listed in Figure 2-18. This XML document creates a single member structure that is constrained on one node, and loaded on the other node as shown in Figure 2-19. Notice that this conforms to the DTD listed above. For example, the DTD declares – “<!ATTLIST units length (meters | centimeters | feet | inches) #REQUIRED ...>” which is satisfied by the text below that says “<units length = "meters" force = "kiloNewtons"></units>”.

```

<StructureXML>

  <units length = "meters" force = "kiloNewtons"></units>

  <uniqueJoints>
    <Joint JointID = "J:0">
      <coordinates x = "0.0" y = "0.0" z = "0.0"></coordinates>
      <constraints Rx = "true" Ry = "true" Rz = "true"
                  Mx = "true" My = "true" Mz = "true"></constraints>
      <loads Fx = "0.0" Fy = "0.0" Fz = "0.0"
            FMx = "0.0" FMy = "0.0" FMz = "0.0"></loads>
    </Joint>
    <Joint JointID = "J:1">
      <coordinates x = "0.0" y = "100.0" z = "0.0"></coordinates>
      <constraints Rx = "false" Ry = "false" Rz = "false"
                  Mx = "false" My = "false" Mz = "false"></constraints>
      <loads Fx = "500.0" Fy = "0.0" Fz = "0.0"
            FMx = "0.0" FMy = "0.0" FMz = "0.0"></loads>
    </Joint>
  </uniqueJoints>

  <members>
    <Member jStart = "0" jEnd = "1">
      <properties A = "0.0634" J = "3.1E-5" Iy = "0.0129" Iz = "5.0E-4"
                  E = "2.07E8" G = "8.01E7" B = "0.0"></properties>
      <memberRelease sMR = "false" eMR = "false"></memberRelease>
    </Member>
  </members>

</StructureXML>

```

Figure 2-18 - StructureXML for a one member Frame

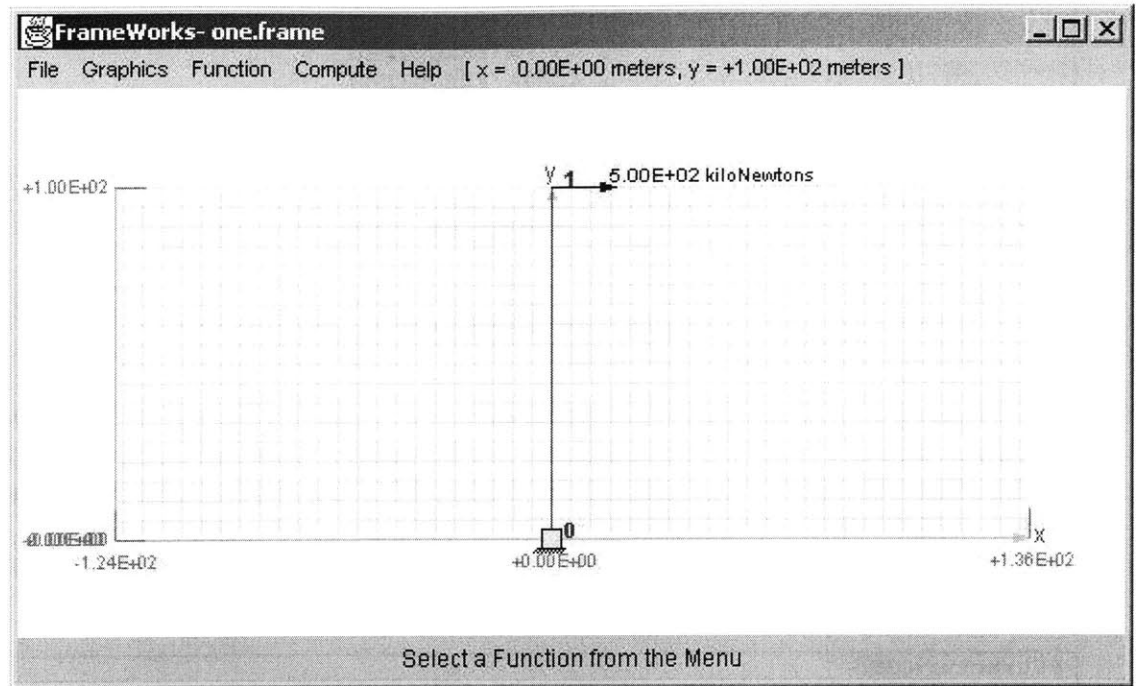


Figure 2-19 - Structure produced from StructureXML using Frameworks

2.5 Open source and it's implications

These programs are released under the GNU General Public License⁵. As such, anyone who wishes to use, and extend Frameworks and Trussworks must also release them under GPL.

Essentially, this makes the software free to share, and ensures that anyone who makes changes and extends the software must also make the source code freely accessible to others. In this way, it is hoped that the software will evolve, and at the same time be accessible to all.

The "program" refers to Frameworks, or Trussworks.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Table 2-11 - The GPL Licence

The preamble of GPL states –

“When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the

⁵ The terms and agreements for GPL, and the full details for this licence can be found at <http://www.gnu.org/copyleft/gpl.html>

original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.”

2.6 A Developer's Perspective

Trussworks and Frameworks have been designed from the ground up to make them run on a wide range of platforms. They do not use “swing” for example, so they work on most computers and browsers that have the Java runtime. They also do not use any proprietary packages. The source code for all classes is freely available so that users can make changes without worrying about having to pay or wait for someone else to make the changes.⁶

The source code for all the classes used in Trussworks and Frameworks are included in the package that contains the executable programs. It is available as a single compressed jar (java archive) file. Refer to the Java documentation to extract the .java files from the jar file.

The package structure is described in Section 2.3.1. There are essentially six packages as shown below.

Package	Description
Graphics	This package contains all the classes that deal with the graphics in the program. To make changes to the gui in the programs, the user needs to make changes to these classes.
Jama	This package does all the matrix computations. To use a different method to solve equations, make changes to the classes included in this package.
Sections	
Support	This package contains all the core classes that define the structure, such as the nodes, and the members.
Tools	Contains classes that perform miscellaneous tasks such as computations, and drawing the structure on the screen.
xml	Handles all the xml parsing for StructureXML.

⁶ Refer to Section 1.1 for more details regarding the licencing.

Table 2-12 - Summary of the package contents

Some potential areas of changes that might prove advantageous are listed below. It also explains briefly what classes would need to be updated to make the changes.

2.6.1 Add more material properties

To add more material properties, simply change the text file called “*materialProperties.txt*”. This file can be found under the Sections package, in the data directory.

The data needs to be formatted such that it contains 4 columns separated by tabs, or spaces. All data is in the SI units. The first column contains the material name, the second column contains the modulus of elasticity (GPa), the third column contains the poisson's ratio, and the last column contains the weight per cubic meter of the material (kN/m^3). An example for stainless steel is shown below in

StainlessSteel	190	0.305	76
----------------	-----	-------	----

Figure 2-20 - material properties text file

2.6.2 Edit or add cross-sections

To add more cross-sections, or edit the cross-sections that appear up in the “*Enter Member Properties*”, the user will need to work with the Sections package. The data for the cross-sections are contained in the data directory, and have a .DAT extension. The “util” sub-package contains all the classes that parse the text files used in the program.

2.6.3 View and edit the structure in 3D in a graphical interface

Currently, Frameworks and Trussworks allow the user to enter the coordinates for the structure using the mouse only in two dimensions. The programs also display the structure in two dimensions only. There are two ways in which a user can extend the programs to view and edit the structure in three dimensions.

1. **Working with objects** – This is the more difficult option of the two, but would be more efficient in the way the program would run. It would require the user to program the graphics package to accept and modify the same objects that Trussworks and Frameworks uses. Refer to the “Support” package for details on the classes that would create the FrameVector, and TrussVector objects.
2. **Working with StructureXML** – Using the StructureXML format, one can easily interface any outside program with Frameworks and Trussworks. Since XML is an open text based system, this can be achieved relatively painlessly. Each program may work with different objects, but as long as they can both understand the XML (by following the DTD discussed in Section 2.4.2), there will be no conflicts.

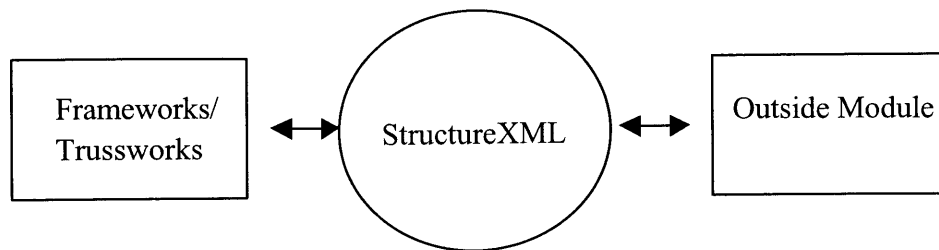


Figure 2-21 - Interfacing with outside programs using XML

The vision is that someone will create a package that will extend the functionality of the programs to three dimensions. The computation package already solves the structure in three dimensions and this would only mean a change in the visualization.

2.6.4 Using a spreadsheet to create the structure

It may sometimes be cumbersome to input the members in Frameworks and Trussworks by clicking the mouse. This would especially be true if the number of members starts getting bigger, or if the structure is in three dimensions.

The programs already accept input in a text format as StructureXML. However, working at the XML level may not always be the most pleasant solution. XML is not designed to be read by users. It would also require the user to be aware of, and understand how the DTD works. For these reasons, if the user decides to input the structure using a text format, an easy solution would be to enter it in a spreadsheet and then export it into StructureXML.

Another possible evolution of the programs therefore may be that someone writes a program that converts text from a spreadsheet into StructureXML. This would let users input the coordinates and member properties of the structure in a spreadsheet format that they are familiar with, and then use the utility to convert it into XML that follows the DTD.

3 Student use

To bring the student into the design process, remote dialogue sessions were run with individual students. To ensure a good record and exchange, the program was modified such that it recorded every mouse click and saved the structure object to a buffer on the server after every click. This allowed the monitoring of the student progress. The students were able to communicate using an instant messaging program to ask questions and make suggestions. Two sample “chat” transcripts are shown below in Table 3-1, and Table 3-2. The set up for this testing is shown below in Figure 3-1.

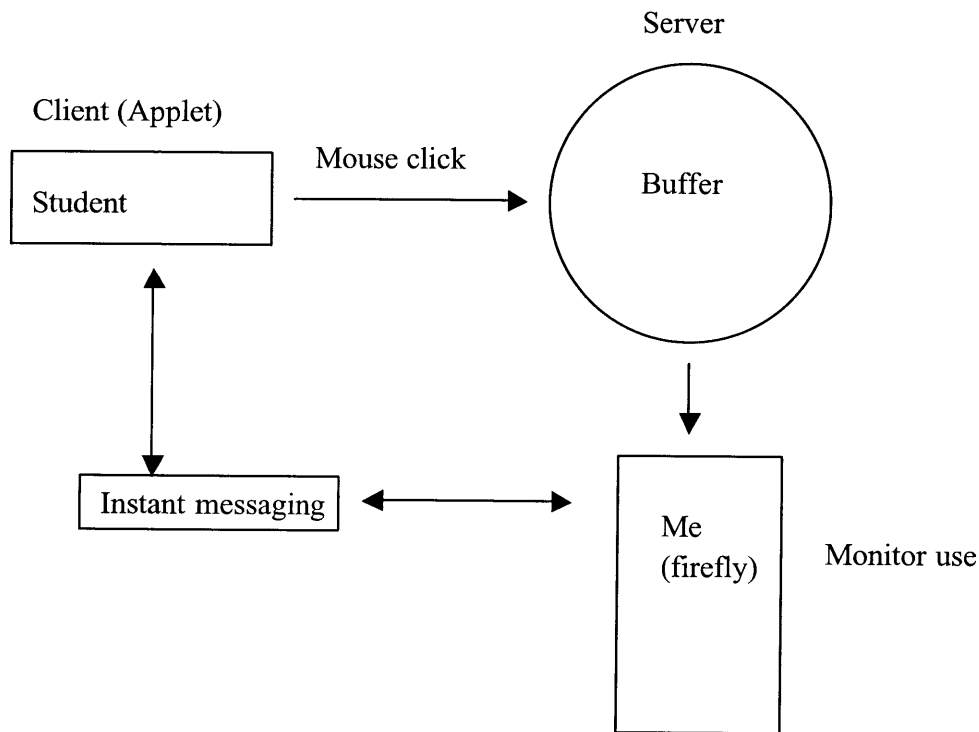


Figure 3-1 - Server Client setup for monitoring student use

The students were directed to the web page shown below in Figure 3-2, where they had to model the truss structure shown.

Thank you for taking the time to test Trussworks

Instructions

- Print this page
- Go to <http://aimexpress.oscar.aol.com/cgi-bin/launch.pl>
- Sign in - Screen Name: TrussWorksTester, Password: trussworks
- I will then send you an instant message, which will open in a new window.

This will be our mode of communication. (If you don't get a message from me at

This will be our mode of communication. **(If you don't get a message from me at this point, I'm not online)**

- Next, go to <http://sthapit.mit.edu/trussworks> and click on the "start program" button. (Mac users: please use Internet Explorer)

- Using the program, *trussworks*, which should be open on your screen, model as best as you can the structure below.

- You can save your progress at anytime using the menu File -> Save to server
- When done, report the following -

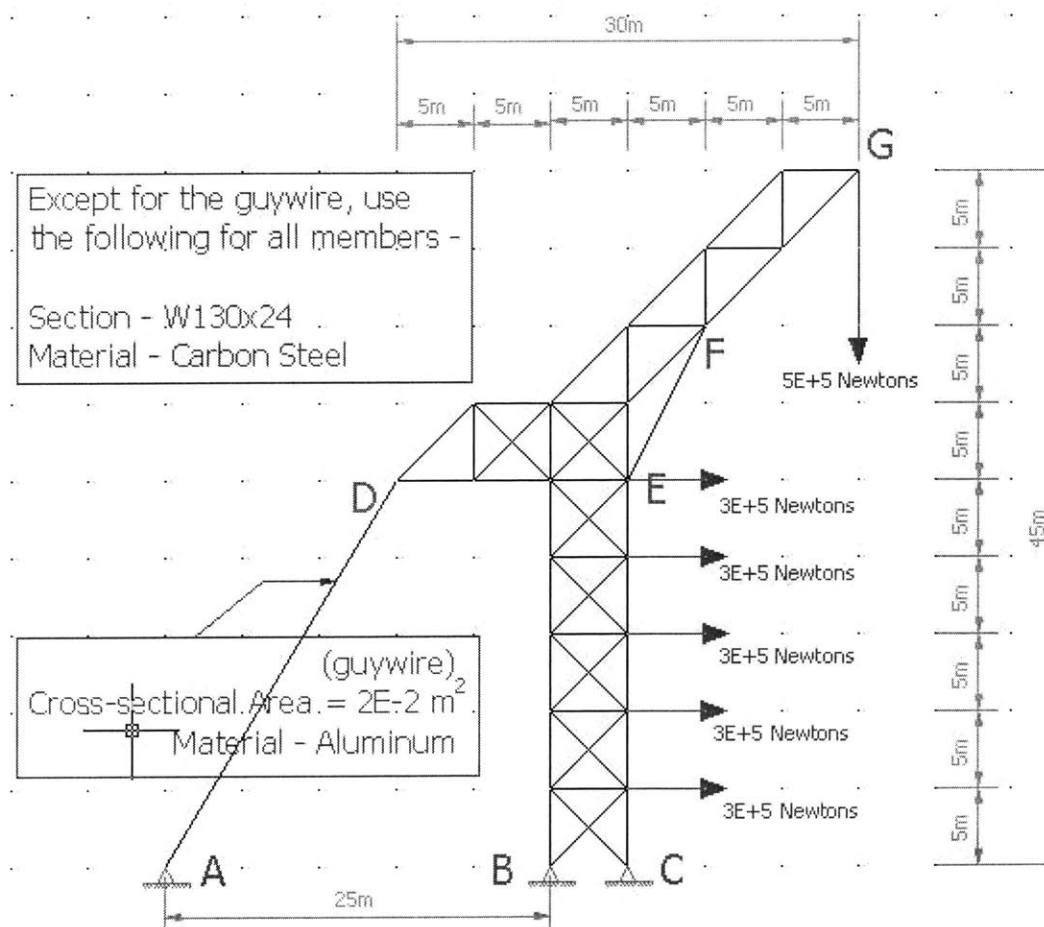
----- Reactions at node B

----- Displacement of node G

----- Member force of member E - F

- Next, remove member A - D, and report the new deflection of node G

- When you encounter any problems, have questions, or any comments, please use the instant messenger to communicate. I will use your feedback to improve the program.



[View trials by other users](#)



Figure 3-2 - User trial screen

<p>TrussWorksTester: Do I have to specify some particular coordinates for the nodes? firefly: No, they're all relative. As long as the lengths of the members are correct, the coordinates doesn't matter. firefly: Don't forget to set extents to set the height/width of your screen to something easy to work with (Graphics -> Set Extents) firefly: also helpful might be to turn on ortho which will only draw vertical and horizontal lines. (either hold the shift key, or go to graphics -> ortho) TrussWorksTester: what is the sign for a roller? firefly: A roller is only constrained in one direction. In this case, you would constrain the node in the Y direction TrussWorksTester: Are the member properties the same for all the members? firefly: yes, apply the same member properties for all members in the structure TrussWorksTester: Can I use exponential values for A and E? or have I to enter values such as 0.00009? firefly: the program accepts exponential format. so .00009 is the same as 9E-5 TrussWorksTester: question1: TrussWorksTester: question1: TrussWorksTester: Rx0 = 2.36E-11N Ry0 = 7.50E+03N TrussWorksTester: Ry4 = 7.49E+03N TrussWorksTester: member force in 1-6: Fx = -6.06E+02N (compression) TrussWorksTester: nodal displacement @ 7: X:+4.88E-7 m; Y: -3.85E-06 m firefly: excellent! many thanks. don't forget to save your structure. File -> Save to Server -> HP.truss TrussWorksTester: done</p>	
<p>User Skill Level:</p> <p>Operating System of user:</p> <p>Summary of User interaction:</p> <p>Author's comments:</p> <p>Changes to program:</p> <p>Visual Source Safe Version:</p>	<ul style="list-style-type: none"> • Civil Engineering graduate student • Windows 2000 Advanced Server • User was familiar with program and had little problem modeling the structure • None • 3

Table 3-1 - Example tester for Trussworks – 1

firefly (7:21:38 PM): hi. please use this window to ask any questions, or post any comments. i will use this information to improve the program

firefly (7:22:12 PM): can you state your operating system, and browser type?

TrussWorksTester (7:22:40 PM): windows 95/netscape

TrussWorksTester (7:23:00 PM): the truss window is not responding

firefly (7:23:28 PM): try minimizing the browser, the dialog box is probably hidden behind the main window

firefly (7:24:12 PM): did that work?

TrussWorksTester (7:26:43 PM): ok

TrussWorksTester (7:27:55 PM): so i have to draw the whole structure, right?

firefly (7:28:10 PM): that is correct.

firefly (7:34:57 PM): you might find it easier if you set the correct grid size, and select "snap to grid"

TrussWorksTester (7:35:46 PM): let me actually try to print this out

firefly (7:35:54 PM): good idea

TrussWorksTester (7:53:54 PM): how do i dra the pins?

TrussWorksTester (7:53:57 PM): draw

firefly (7:54:10 PM): Function -> Constrain Nodes

firefly (7:54:59 PM): your tower is a little taller than the exercise, you might want to delete some members

TrussWorksTester (7:55:23 PM): oh yeah

TrussWorksTester (7:55:24 PM): hehe

firefly (7:55:41 PM): good job so far though

TrussWorksTester (8:01:15 PM): how do i select what node i want to put the load on?

firefly (8:01:35 PM): Function -> Load node, then select node by clicking on them

TrussWorksTester (8:02:55 PM): is there an undo?

firefly (8:03:12 PM): no, no undo in the program.

firefly (8:03:44 PM): you'll have to remove members, and add them again

TrussWorksTester (8:04:00 PM): ok

firefly (8:05:16 PM): you have a slight problem

firefly (8:05:43 PM): seems like you entered a member on the same joint somehow

TrussWorksTester (8:06:39 PM): what do you mean?

TrussWorksTester (8:06:47 PM): like node 91?

firefly (8:06:59 PM): yeah, i'm not sure how that happened

firefly (8:07:50 PM): try this -> download from server, buffer.truss

TrussWorksTester (8:08:33 PM): ok

TrussWorksTester (8:08:38 PM): now what?

firefly (8:08:48 PM): did you get the structure on the screen?

TrussWorksTester (8:08:53 PM): yeah

firefly (8:09:02 PM): continue with the exercise please

TrussWorksTester (8:09:09 PM): ok

TrussWorksTester (8:12:26 PM): i've constrained in the x and y directions.

firefly (8:12:31 PM): good

TrussWorksTester (8:15:17 PM): how do i enter the cross-sectional area for the members?

firefly (8:15:31 PM): Function -> Enter Member Properties

TrussWorksTester (8:15:39 PM): yes
firefly (8:16:12 PM): did that work?
TrussWorksTester (8:16:22 PM): ok, what is the cross-sectional area for the members (W130X24)?
firefly (8:16:45 PM): select it on the drop down menu on the right of the window (says choose a section)
firefly (8:17:12 PM): did that work?
TrussWorksTester (8:17:21 PM): ah no wonder
TrussWorksTester (8:17:47 PM): when it first popped up the it was already on "W" but nothing showed
TrussWorksTester (8:17:57 PM): had to actually select it again
firefly (8:17:57 PM): good point. i will change that in the program.
firefly (8:18:28 PM): this is why i'm doing this - i need feedback from users like you!
TrussWorksTester (8:19:35 PM): what's the selection for guywire?
firefly (8:19:54 PM): are you asking how you select that member?
TrussWorksTester (8:20:09 PM): in the member properties
TrussWorksTester (8:20:31 PM): i need to put "W" or "S" or...
firefly (8:20:58 PM): enter the cross-sectional area yourself, no need to select a cross-section. put in $2e-2$
TrussWorksTester (8:21:20 PM): ok, so i can leave it on whatever on the right?
firefly (8:21:27 PM): that is correct
TrussWorksTester (8:21:38 PM): ok
firefly (8:21:53 PM): that is a good point - that is a possible confusion. another change.
TrussWorksTester (8:23:08 PM): do i need to draw the pins?
firefly (8:23:20 PM): yes. Function -> Constrain nodes
TrussWorksTester (8:24:23 PM): i didn't know i had to select while that "constrain node" pop-up was up
firefly (8:25:08 PM): ok. so the message at the bottom "click on node[s] to Constrain" was not clear?
firefly (8:25:28 PM): where would be a good place to ask the user what to do?
TrussWorksTester (8:26:00 PM): hehe, i didn't notice that at first but on the pop up menu wouldn't be a bad idea
firefly (8:26:23 PM): excellent suggestion
firefly (8:26:33 PM): will do that on the next version for sure
TrussWorksTester (8:28:12 PM): i've selected "reactions". then what?
firefly (8:28:44 PM): you should see the reactions displayed on the screen. in your case, you didn't constrain the nodes in the z direction, which makes the structure unstable.
firefly (8:29:08 PM): so, constrain nodes 23, 0, and 6, in x, y, as well as in z
TrussWorksTester (8:29:23 PM): oh ok- makes sense.
TrussWorksTester (8:29:29 PM): only those three?
firefly (8:29:39 PM): yes
firefly (8:31:50 PM): you also have not set the member properties for the guy wire correctly. can you do that again please?
TrussWorksTester (8:31:56 PM): ok
TrussWorksTester (8:33:39 PM): how about now?
firefly (8:34:05 PM): no, it's still the same

firefly (8:34:15 PM): can you tell me what you did? the exact steps
firefly (8:35:25 PM): there you go
firefly (8:35:35 PM): what problem were you having the first time?
TrussWorksTester (8:35:50 PM): i don't know
TrussWorksTester (8:36:12 PM): i don't think it was the guywire
firefly (8:36:18 PM): well, it looks good now. can you report what's asked please?
TrussWorksTester (8:36:38 PM): the member for EF was wrong
TrussWorksTester (8:36:59 PM): ok one sec.
TrussWorksTester (8:37:59 PM): how do i get the displacement value at G, or do i need to actually measure that?
firefly (8:38:11 PM): no, use the program to give you that number
TrussWorksTester (8:40:10 PM): reaction B: 3.87e5 N to the left (negative) in the 'x' direction. And 1.21e6 N down in the 'y' (negative) direction
firefly (8:41:33 PM): correct. what about the displacements?
firefly (8:41:42 PM): and member forces
TrussWorksTester (8:41:54 PM): i'm trying to get the displacement
TrussWorksTester (8:42:13 PM): i can only get the picture. how do i get the value?
firefly (8:42:22 PM): Compute -> Text Output
TrussWorksTester (8:43:45 PM): Nodal displacement at G:+0.36E-01 -0.29E-01 0.00E+00
firefly (8:44:07 PM): good job yogi
TrussWorksTester (8:44:10 PM): in the x, y, and z directions, respectively
TrussWorksTester (8:45:13 PM): force on member EF:
TrussWorksTester (8:45:33 PM): -1.18e6 N (compression)
firefly (8:45:39 PM): correct
TrussWorksTester (8:47:18 PM): after removing member AD, the nodal displacement at 'G' is
TrussWorksTester (8:47:20 PM): +2.20E+00 -1.23E+00 0.00E+00
firefly (8:47:27 PM): excellent!

-
- User Skill Level:**
 - Aerospace Engineering undergraduate student
- Operating System of user:**
 - Windows 95, running Netscape
- Summary of User interaction:**
 - User had problem of dialog box "hiding" behind main window, which made it appear as if the program had frozen
- Author's comments:**
 -
- Changes to program:**
 - Removed problem of multiple nodes at same point when editing node coordinates in snap mode

Changes to program:	<ul style="list-style-type: none"> Removed problem of multiple nodes at same point when editing node coordinates in snap mode
Visual Source Safe Version:	<ul style="list-style-type: none"> 12

Table 3-2 - Example tester for Trussworks - 2

Visual SourceSafe was used for version control for the source code. After each user trial, the programs were modified to reflect changes that were thought to best address the problems the students had. To save space, all chat sessions have not been included, but the changes made to the program after each trial is summarized below in Table 3-3.

User Skill Level:	<ul style="list-style-type: none"> Civil Engineering graduate student
Operating System of user:	<ul style="list-style-type: none"> Windows 2000 Advanced Server
Summary of User interaction:	<ul style="list-style-type: none"> User was familiar with program and had little problem modeling the structure
Author's comments:	<ul style="list-style-type: none"> None
Changes to program:	<ul style="list-style-type: none"> None
Visual Source Safe Version:	<ul style="list-style-type: none"> 3
User Skill Level:	<ul style="list-style-type: none"> Civil Engineering undergraduate student
Operating System of user:	<ul style="list-style-type: none"> Windows 98
Summary of User interaction:	<ul style="list-style-type: none"> User did not know how to start Found a bug which reduces "numJoints" in program if you hit esc key while adding members <ul style="list-style-type: none"> User wanted to know how to enter exact coordinates while clicking and dragging
Author's comments:	<ul style="list-style-type: none"> Found some consistent problems, namely not knowing how to start
Changes to program:	<ul style="list-style-type: none"> Fixed bug mentioned above Changed "set extents" to "Set Screen Dimensions" Made the "set Screen Dimensions" appear by default on starting the program, and when selecting "File -> New"
Visual Source Safe Version:	<ul style="list-style-type: none"> 6

<p>User Skill Level:</p> <p>Operating System of user:</p> <p>Summary of User interaction:</p> <p>Author's comments:</p> <p>Changes to program:</p> <p>Visual Source Safe Version:</p>	<ul style="list-style-type: none"> • Civil Engineering graduate student • Windows 2000 Professional • User had same problem of "load node" dialog boxes disappearing behind the main window <ul style="list-style-type: none"> • Had to keep telling user to save file so that his progress could be monitored • Consider changing Dialog[boxes] to Frames. This will alleviate problem of load, and constrain dialog boxes "disappearing" [behind main screen] when selecting nodes by showing up on the task bar below (on windows machines) • set boolean, which if set to true, will save a buffer file, "buffer.truss" everytime the user clicks the mouse <ul style="list-style-type: none"> • (Working on) allow user to explicitly set grid size • 7
<p>User Skill Level:</p> <p>Operating System of user:</p> <p>Summary of User interaction:</p> <p>Author's comments:</p> <p>Changes to program:</p> <p>Visual Source Safe Version:</p>	<ul style="list-style-type: none"> • Civil Engineering undergraduate student • Unknown • The exponential/scientific format for viewing numbers was uncomfortable to user <ul style="list-style-type: none"> • There were problems with changing the node coordinates on the user's side • User logged off before completion of exercise • The "save buffer" feature worked very well, which made it possible to monitor the user's progress without having to prompt him to save. • User is prompted in the beginning to set screen size, and also the size of the grid <ul style="list-style-type: none"> • Program can also now let the user set the grid size explicitly, and snap to grid works as well • 10

User Skill Level:	<ul style="list-style-type: none"> • Civil Engineering graduate student
Operating System of user:	<ul style="list-style-type: none"> • Windows 2000 (Professional), running Netscape
Summary of User interaction:	<ul style="list-style-type: none"> • The user was extremely quick to learn the program and modeled the structure with little problem • User was a little confused with the dialog box "disappearing" when selecting nodes
Author's comments:	<ul style="list-style-type: none"> • User later let me know that he would have liked to see more advanced features like in AutoCad, where you can right click on a member to edit it's properties
Changes to program:	<ul style="list-style-type: none"> • None
Visual Source Safe Version:	<ul style="list-style-type: none"> • 10
User Skill Level:	<ul style="list-style-type: none"> • Civil Engineering graduate student
Operating System of user:	<ul style="list-style-type: none"> • Windows 98, running Internet Explorer
Summary of User interaction:	<ul style="list-style-type: none"> • User had same problem of dialog box disappearing behind the main window • Suggested that a copy and paste feature would be helpful • Grid lines seemed too light on users laptop • User suggested to use icons instead of checking x,y,z for constraining nodes • User said she was used to having a "solve" button to recompute results to make her feel like the program was computing. The current model made her doubt if anything was going on when selecting a menu item • Animations would be nice, according to user
Author's comments:	<ul style="list-style-type: none"> •
Changes to program:	<ul style="list-style-type: none"> • Modified Program to change Load and Constrain Dialog boxes into Frames, which should make alleviate problem of the dialog box diappearing when selecting nodes
Visual Source Safe Version:	<ul style="list-style-type: none"> • 11

User Skill Level:	<ul style="list-style-type: none"> • Aerospace Engineering undergraduate student
Operating System of user:	<ul style="list-style-type: none"> • Windows 95, running Netscape
Summary of User interaction:	<ul style="list-style-type: none"> • User had problem of dialog box "hiding" behind main window, which made it appear as if the program had frozen • Asked if there was an "undo" function available • In edit member properties window, it seemed like the W sections were already loaded, when it hadn't been loaded yet • Said it might be more clear if in the constain, and load windows, there were some indication that the nodes had to be selected first before applying them
Author's comments:	<ul style="list-style-type: none"> •
Changes to program:	<ul style="list-style-type: none"> • Removed problem of multiple nodes at same point when editing node coordinates in snap mode
Visual Source Safe Version:	<ul style="list-style-type: none"> • 12

Table 3-3 - Summary of user interaction and changes made to the program

4 Conclusions

Frameworks and Trussworks have been programmed in the Java environment, which is relatively robust. Extensive user trials were conducted to test for any flaws in the programs. Programming in an object-oriented language also carries the benefit that classes can be easily modified. Care was taken so that the package structure is constructed to make it easy to pinpoint which part of the code does what.

The programs are also free, and readily available on the Internet. To make is as accessible as possible, they can be run as applets on any web browser, or downloaded and run as an application. Since the programs were designed with students in mind, they are easy to learn, and hopefully fun to use as well.

5 References

- [Bathe 96] Bathe, K., 1996, "Finite Element Procedures".
- [Bucciarelli 00] Bucciarelli, L. L., 2000, "Engineering Mechanics for Structures", pp. 219-223.
- [Campione 00] Campione, M., Walrath, K., Huml, A., "The Java™ Tutorial: A Short Course on the Basics". Also available online as "The Java Tutorial: A practical guide for programmers", <<http://java.sun.com/tutorial>>
- [Connolly 01] Connolly, D., "Extensible Markup Language (XML)", 14 November 2001, < <http://www.w3.org/XML>>
- [Kassimali 99] Kassimali, A., 1999, "Matrix Analysis of Structures".
- [Kausel, Roesset 00] Kausel, E., Roesset, J.M., 2000, "Advanced Structural Dynamics", pp. 25-30
- [Martin 96] Martin, H.C., 1996, "Introduction to Matrix Methods of Structural Analysis".
- [Naughton 96] Naughton, P., 1996, "The JAVA Handbook", pp. 391-409.
- [Schildt 01] Schildt, H., 2001, "Java 2: The Complete Reference", Fourth Edition.
- [White 00] White, J., 2000, "Introduction to Numerical Simulation", Class Notes for course 6.336.
- [Will 99] Will, K.M., 1999, "Structural Analysis II", Class Notes for course CE 4550.
- [Young 00] Young, M.J., 2000, "Step by Step XML".

6 Appendix

6.1 Tutorial and Demo of a sample problem

Before a computer program can analyze a structure, it must first be appropriately modeled (Section 1.4). This section takes you through the steps necessary to model a sample structure using the graphical interface⁷, to input the necessary loading conditions and constraints, and to display the forces and deformations computed.

6.1.1 Problem Statement

Using Frameworks, model the structure shown in Figure 6-1. All members are made of carbon steel, and use the W130x24 cross-section. Report the deflections at node A, the reactions and node D, and the member forces in node C. The circles in the members in the middle of the structure indicate moment releases.

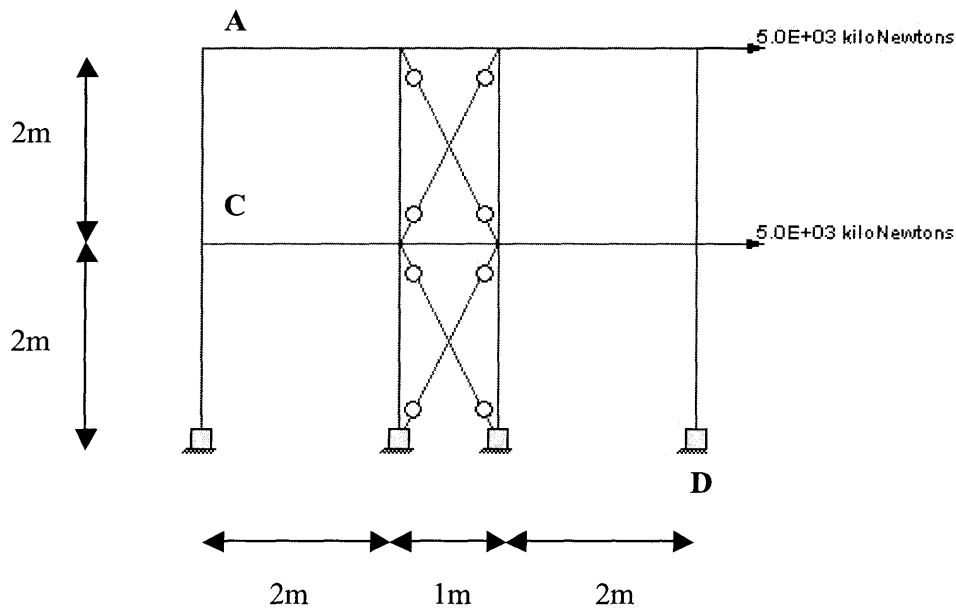


Figure 6-1 - Sample Frame to be analyzed

6.1.2 Set Units

Start the program, Frameworks (Section 6.4). The first dialog box that appears is shown in Figure 6-2. Pick the “*KiloNewtons/meters*” option, and input a Length of 20 meters, and grid spacing (snap resolution) of 2 meters.⁸

⁷ To create a structure using a text based input, use StructureXML. See Section 2.4.2.

⁸ All of these initial choices can be changed later in the program. See Section 6.2.2.

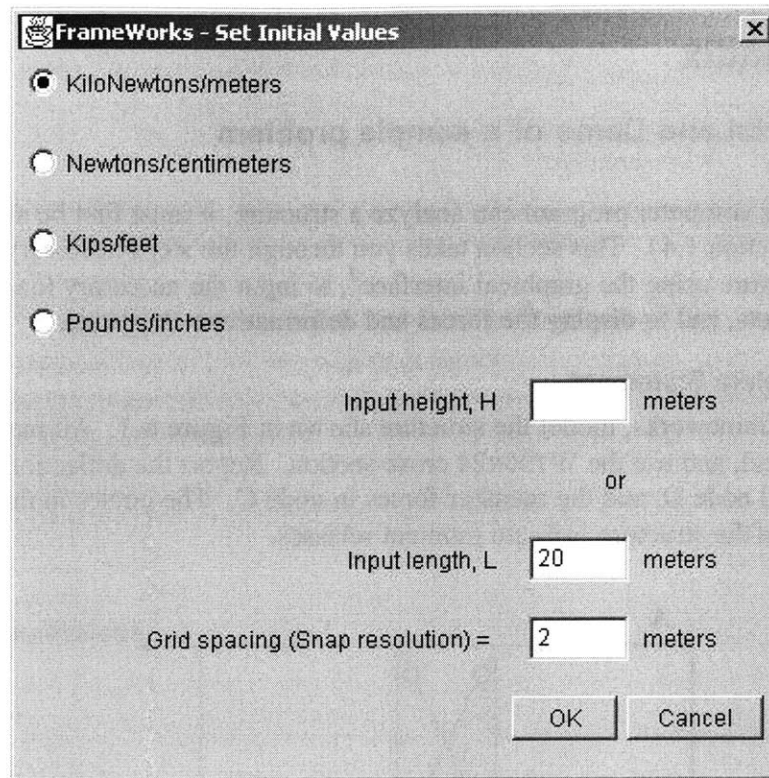


Figure 6-2 - Dialog box to set initial values and units

6.1.3 Create the structure

Go to the “*Function*” menu, and select “*Add Member*” as shown in Figure 6-3. Click once to add the first node of the member, move the mouse, and click again for the second node.⁹ Repeat the process until you have a structure like that shown in Figure 6-4.

⁹ Members can be removed by selecting “*Remove member*” from the “*Function*” menu, and node coordinates can be changed later by selecting “*Edit Node Coordinates*” from the “*Function*” menu.

The program is set to snap to the grid spacing specified above (2 meters) by default. To prevent it from snapping to the grid, go to the “*Graphics*” menu, and deselect “*Snap to Grid*”.

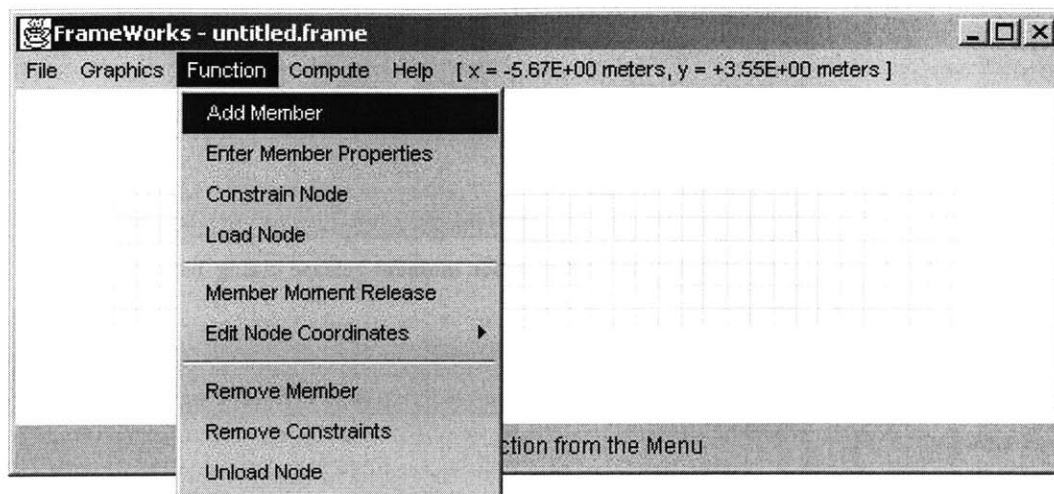


Figure 6-3 - Add Member

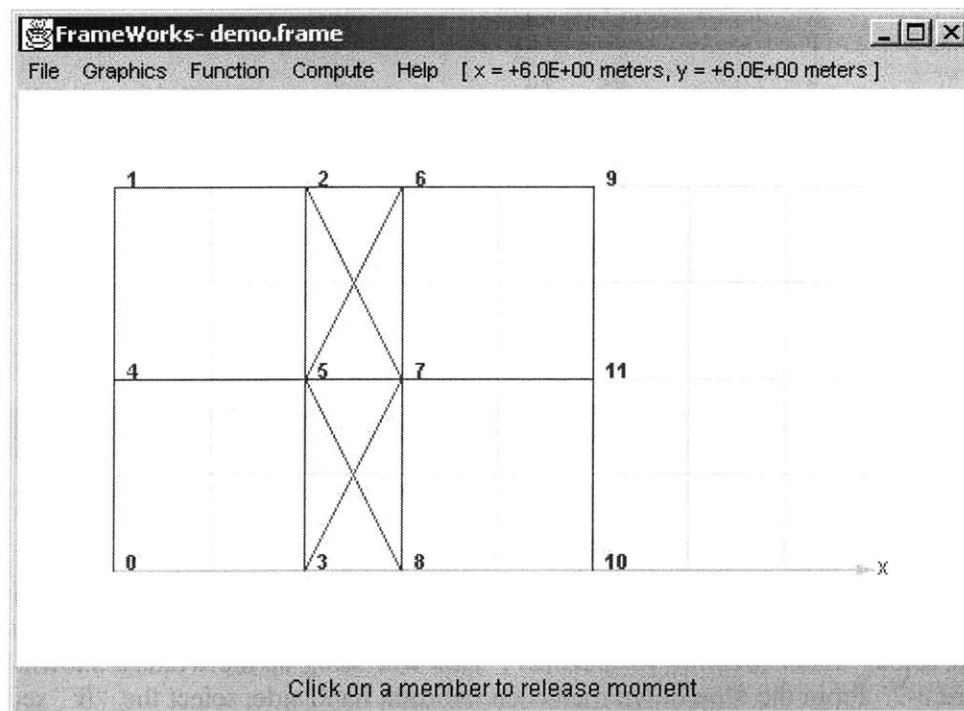


Figure 6-4 - Members added

Next release the internal moments in the members in the middle of the structure as shown in Figure 6-1. Select "Member Moment Release" from the "Function" menu, and click on the members to release moments. This will bring up the dialog-box shown in Figure 6-5. Release the moments at the appropriate nodes, and click on the "Release" button. When done, the structure will look like that shown in Figure 6-6.

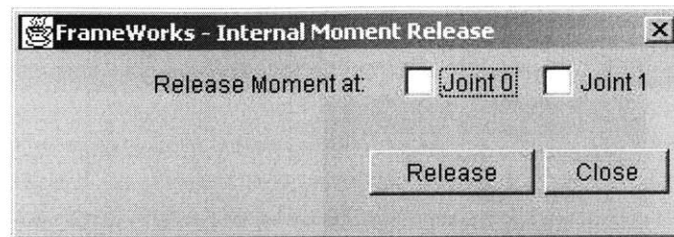


Figure 6-5 - Internal member moment release dialog-box

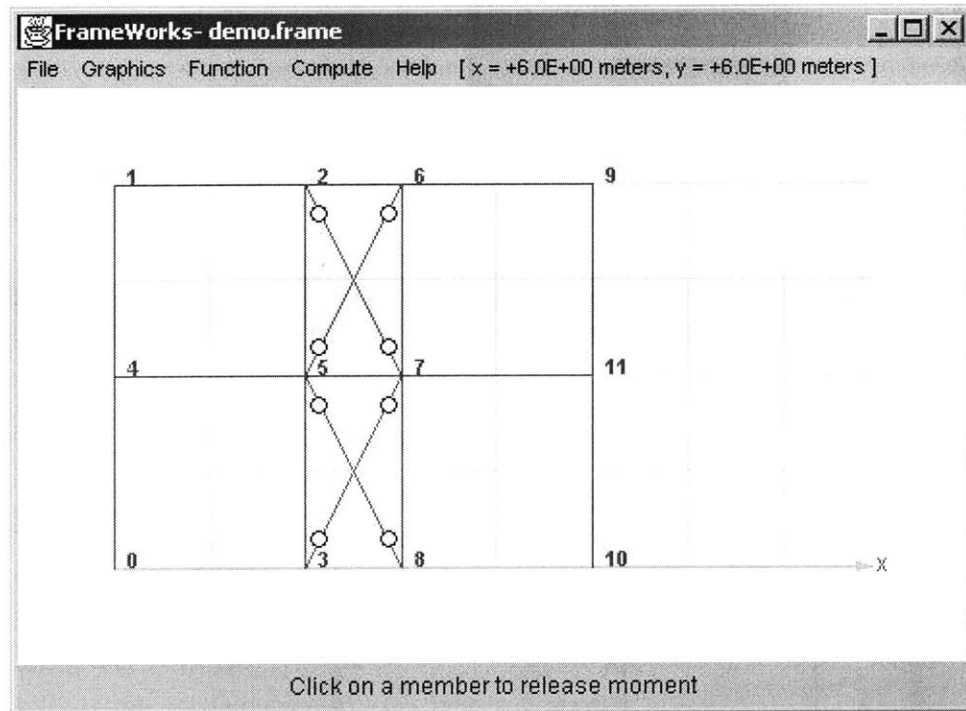


Figure 6-6 - Structure after moments have been released

6.1.4 Enter member properties

Once all the members have been added, enter their member properties. In this example, all members are W130x24 and made of carbon steel. From the *Function* menu, select *Enter Member Properties*. This will bring up the window shown in Figure 6-7. From the drop down menus on the right hand side, select the *W* section, and then select *W130x24*. Next, select *CarbonSteel* from the drop down menu on the bottom right. Select the *Apply to all members* option (default option), and click on the *Apply* button.

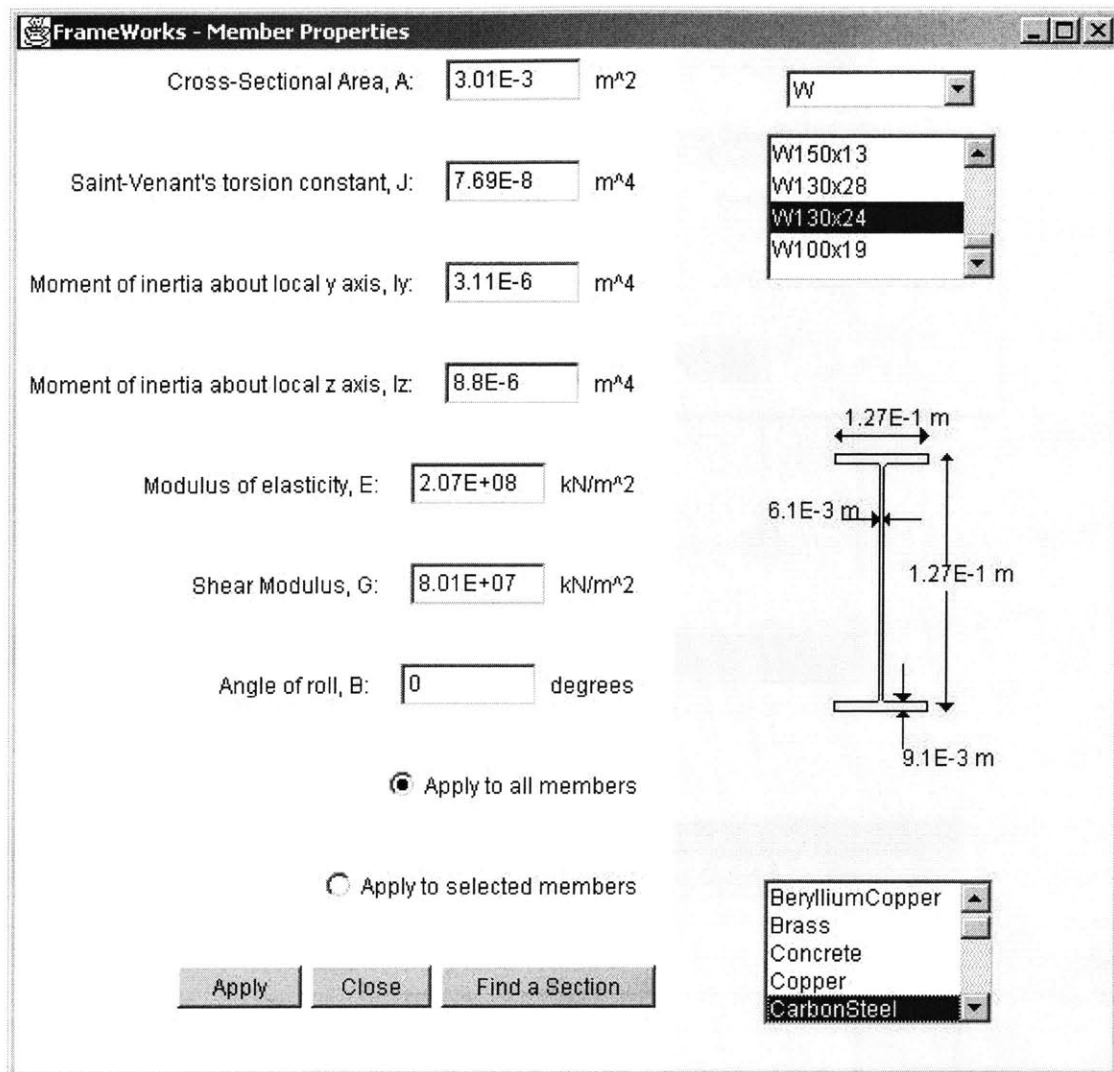


Figure 6-7 - Window to enter Member Properties

6.1.5 Constrain Node

To constrain a node, select "Constrain Node" from the "Function" menu. This will bring up another window as shown in Figure 6-8. Next, click on the nodes to be constrained on the main screen. Selected nodes will be highlighted in blue. Then check the directions the nodes to be constrained, (in this example, check all displacements and rotations), and click on the "Constrain" button. The constrained nodes should be drawn in yellow, as shown in Figure 6-9.

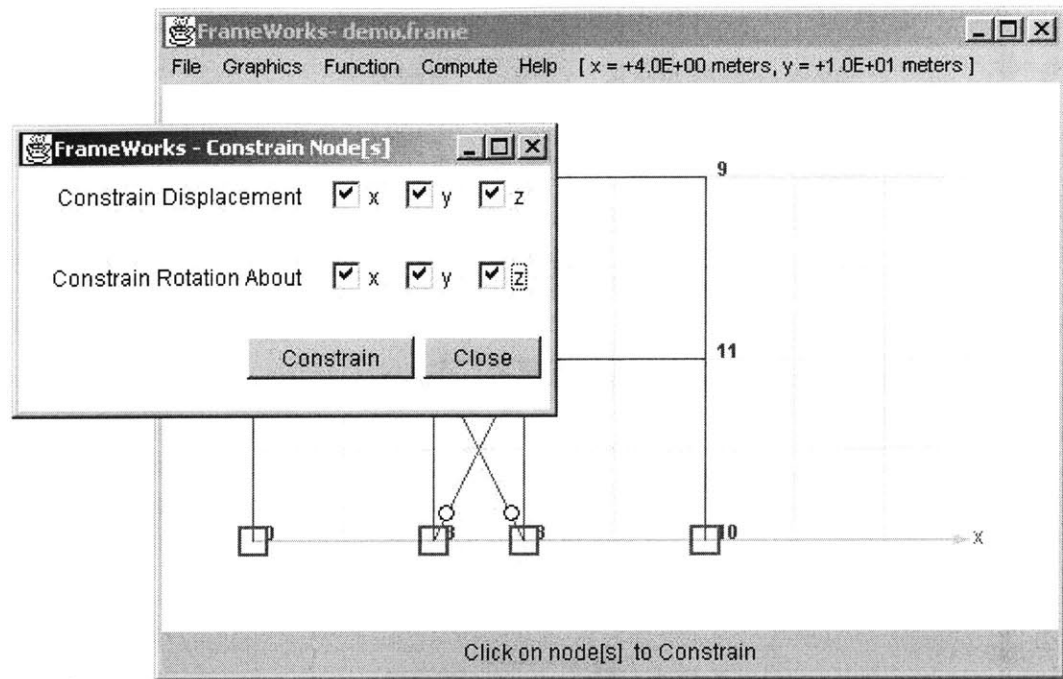


Figure 6-8 - Constrain Node Window

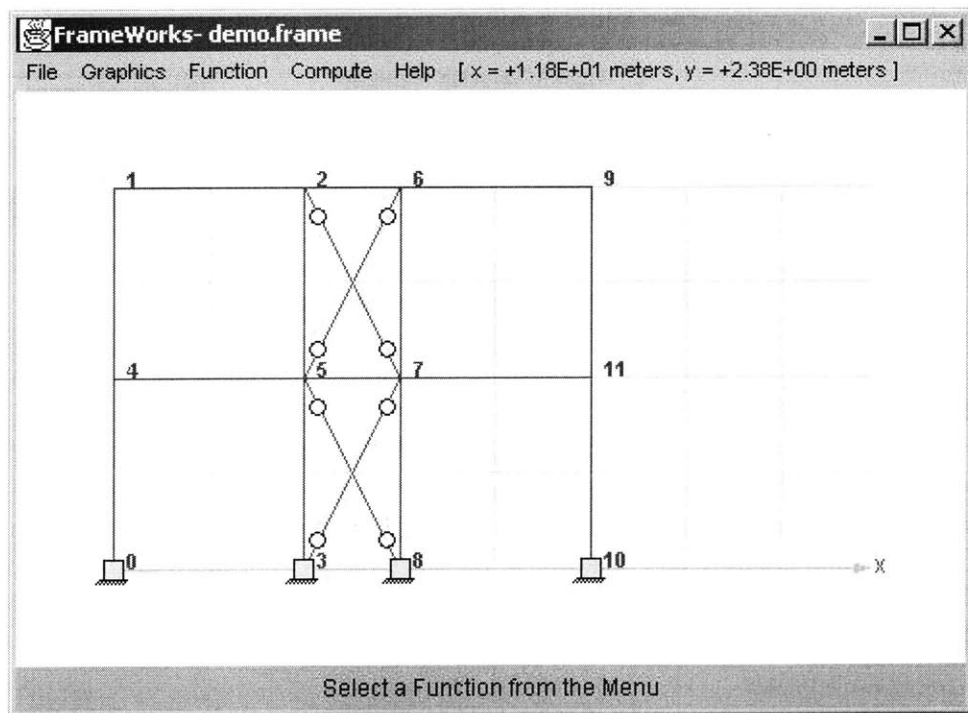


Figure 6-9 - Structure after nodes have been constrained

6.1.6 Load Node

To apply a load on a node, go to the “Function” menu, and select “Load Node”. This will bring up the window shown in Figure 6-10. Next, go back to the main window and click on the nodes to be loaded. Selected nodes appear highlighted in blue. Then input the loads to be applied (in our case 5000 kN in the positive x direction), and click on the “Load” button. The nodal loads should appear on the main screen as shown in Figure 6-11.

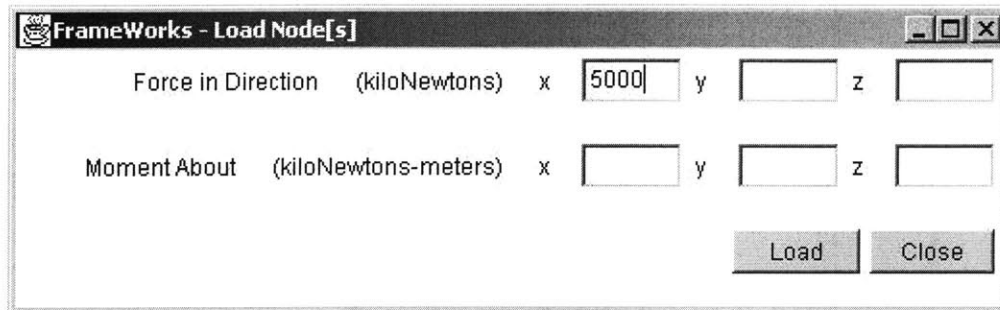


Figure 6-10 - Load Node window

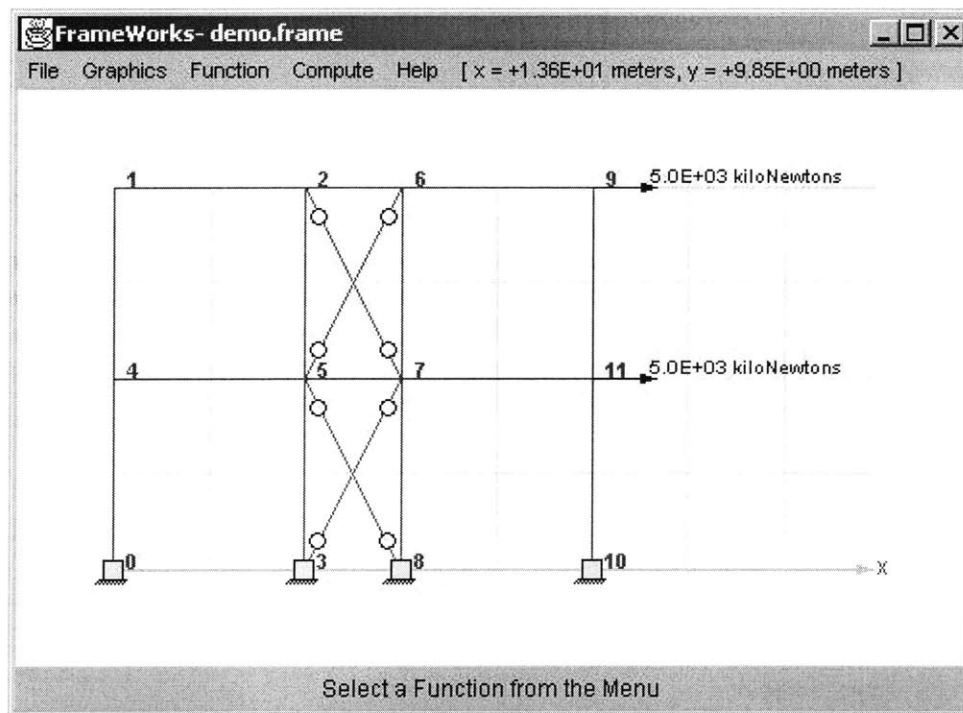


Figure 6-11 - Main Screen after loads have been applied

6.1.7 View Displaced structure

To view the displayed structure due to the applied loads (Section 6.1.6), select “View Displayed Structure” from the “Compute” menu. Different magnification factors may be chosen to properly view the displayed structure. In this case, select “View Displayed

Structure → *x1000*". This will show the displacements magnified by a factor of 1000, shown below in Figure 6-12.

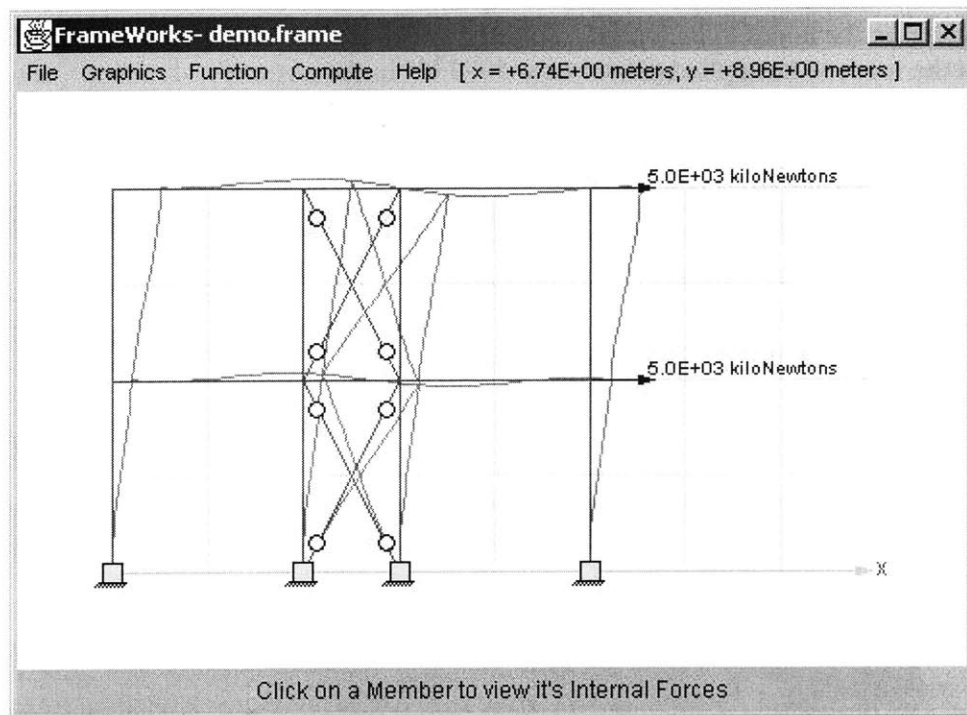


Figure 6-12 - Displaced Structure

6.1.8 View Member Forces

To view the forces developed in the members as a result of the loads applied, select "*Member Forces*" from the "*Compute*" menu. Click on a member to view its forces, and a new dialog-box will pop up showing the member forces (Figure 6-13). In our case, click on member 1-4 (or member A-C as shown in Figure 6-1). The sign convention used is the right hand rule, the positive x direction runs from the start node to the end node. The positive force directions are displayed as black arrows in Figure 6-13.

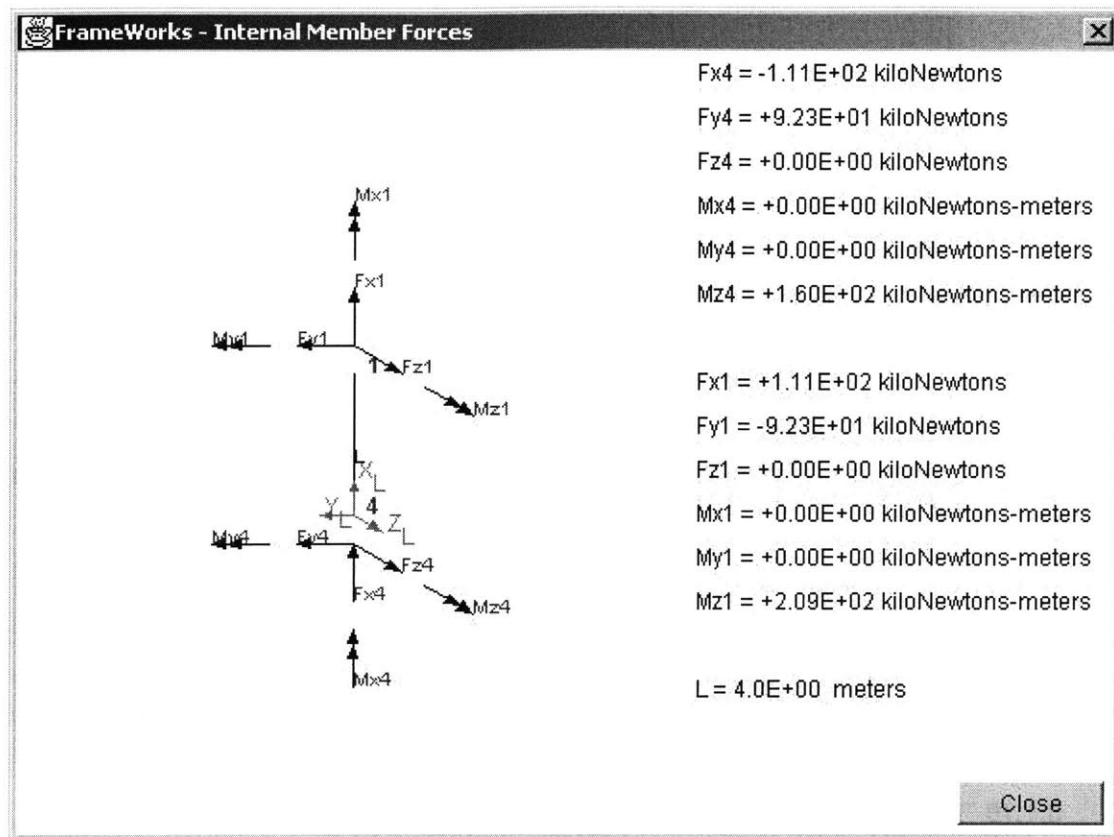


Figure 6-13 - Member Forces

6.1.9 View Reactions

To view the reactions at the constrained nodes (Section 6.1.5) due to the applied loads (Section 6.1.6), select "Reactions" from the "Compute" menu. This will compute the reactions and display them on the main screen. If the numbers look crowded, use the "Zoom \rightarrow Window" option from the "Graphics" menu, and zoom in on the node. In this case, zoom in on node 10 (or node D in Figure 6-1). This will change the main screen to display the structure as shown in Figure 6-14.

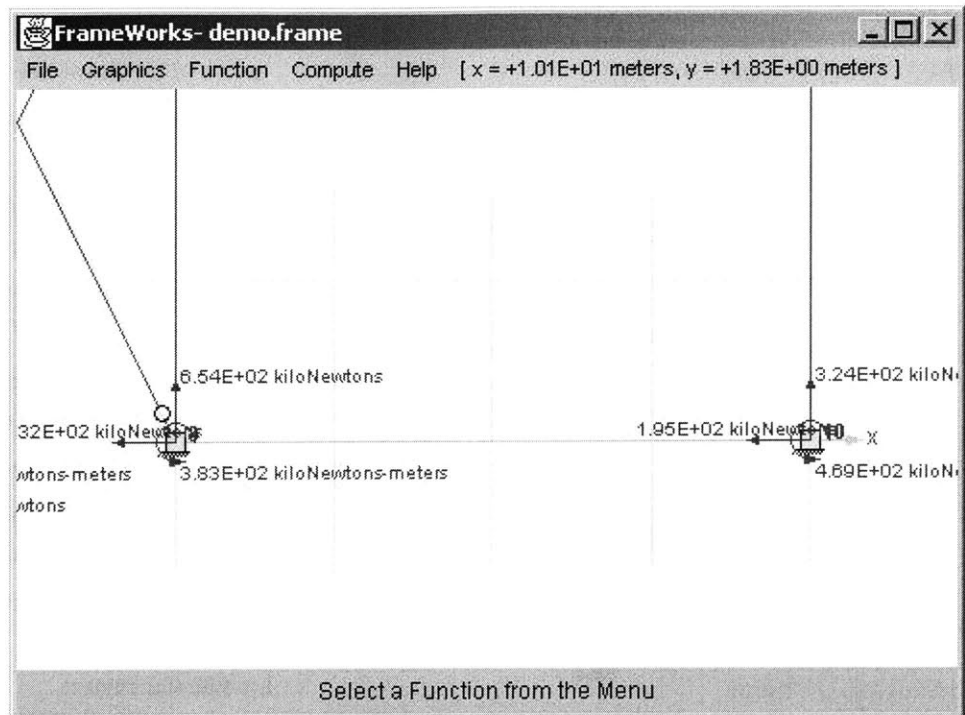
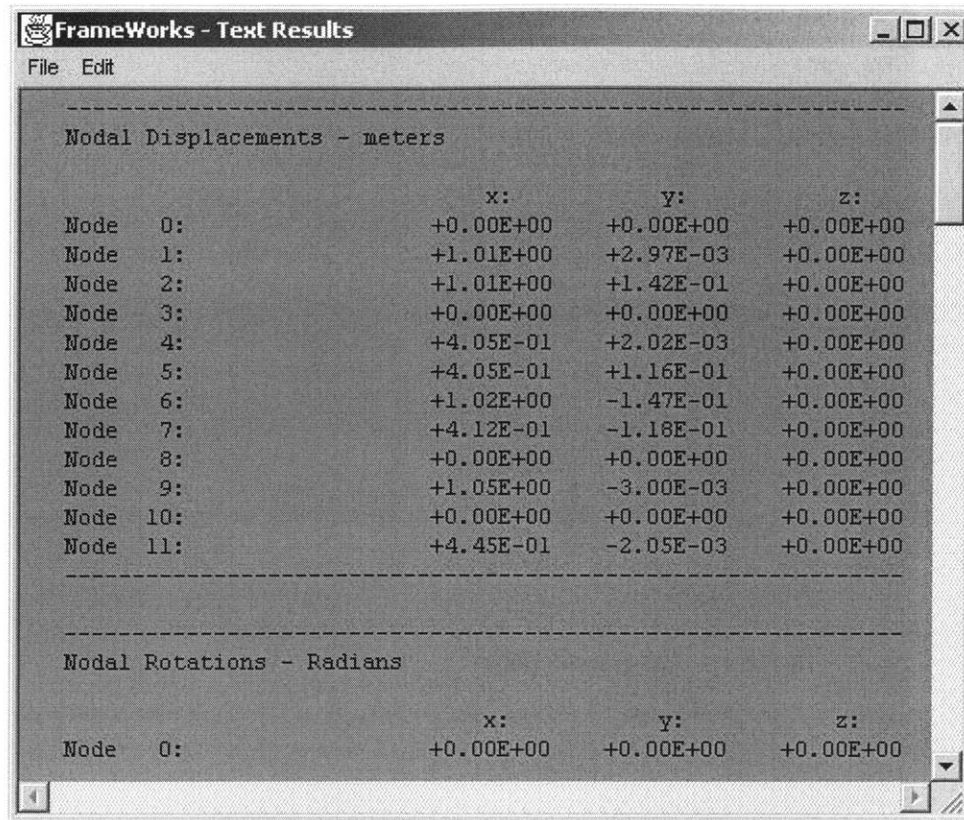


Figure 6-14 - Reactions on the structure

6.1.10 Text Output

To view the results of the computations in a text format, select "*Text Output*" from the "*Compute*" menu. This will bring up a window with the text output of the computations as shown in Figure 6-15. If running the programs as an *application*, select "*Save As...*" from the "*File*" menu in the text window, which will allow the user to save the text output on the local hard-drive. If running the programs as an *applet*, highlight and copy all the text from the text area, and paste it in any text editor. This will allow the results to be saved on the local machine.



The screenshot shows a text window titled "FrameWorks - Text Results" with a menu bar containing "File" and "Edit". The window displays two sections of text output. The first section is titled "Nodal Displacements - meters" and lists data for nodes 0 through 11. The second section is titled "Nodal Rotations - Radians" and shows data for Node 0. The data is presented in a tabular format with columns for node numbers and displacement/rotation values in x, y, and z directions.

Nodal Displacements - meters				
		x:	y:	z:
Node	0:	+0.00E+00	+0.00E+00	+0.00E+00
Node	1:	+1.01E+00	+2.97E-03	+0.00E+00
Node	2:	+1.01E+00	+1.42E-01	+0.00E+00
Node	3:	+0.00E+00	+0.00E+00	+0.00E+00
Node	4:	+4.05E-01	+2.02E-03	+0.00E+00
Node	5:	+4.05E-01	+1.16E-01	+0.00E+00
Node	6:	+1.02E+00	-1.47E-01	+0.00E+00
Node	7:	+4.12E-01	-1.18E-01	+0.00E+00
Node	8:	+0.00E+00	+0.00E+00	+0.00E+00
Node	9:	+1.05E+00	-3.00E-03	+0.00E+00
Node	10:	+0.00E+00	+0.00E+00	+0.00E+00
Node	11:	+4.45E-01	-2.05E-03	+0.00E+00

Nodal Rotations - Radians				
		x:	y:	z:
Node	0:	+0.00E+00	+0.00E+00	+0.00E+00

Figure 6-15 - Text output of calculations

6.1.11 View Stiffness Matrix

To view the stiffness matrix of the structure (Section 1.1), select "*View Stiffness Matrix*" from the "*Compute*" menu. This will show the stiffness matrix of the structure in a text window. Figure 6-16 shows the stiffness matrix for our sample structure.

The screenshot shows a window titled "FrameWorks - Stiffness Matrix" with a menu bar containing "File" and "Edit". The main area displays a table with 23 rows and 5 columns of numerical values in scientific notation. The values are as follows:

1:	+1.56E+05	+0.00E+00	+0.00E+00	+0.00E+00	+
2:	+0.00E+00	+1.56E+05	+0.00E+00	+0.00E+00	+
3:	+0.00E+00	+0.00E+00	+2.41E+02	-2.41E+02	-
4:	+0.00E+00	+0.00E+00	-2.41E+02	+6.45E+02	+
5:	+0.00E+00	+0.00E+00	-2.41E+02	+0.00E+00	+
6:	+6.83E+02	+6.83E+02	+0.00E+00	+0.00E+00	+
7:	-1.55E+05	+0.00E+00	+0.00E+00	+0.00E+00	+
8:	+0.00E+00	-3.41E+02	+0.00E+00	+0.00E+00	+
9:	+0.00E+00	+0.00E+00	-1.20E+02	+0.00E+00	+
10:	+0.00E+00	+0.00E+00	+0.00E+00	-1.53E+00	+
11:	+0.00E+00	+0.00E+00	-2.41E+02	+0.00E+00	+
12:	+0.00E+00	+6.83E+02	+0.00E+00	+0.00E+00	+
13:	-3.41E+02	+0.00E+00	+0.00E+00	+0.00E+00	+
14:	+0.00E+00	-1.55E+05	+0.00E+00	+0.00E+00	+
15:	+0.00E+00	+0.00E+00	-1.20E+02	+2.41E+02	+
16:	+0.00E+00	+0.00E+00	-2.41E+02	+3.21E+02	+
17:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	-
18:	+6.83E+02	+0.00E+00	+0.00E+00	+0.00E+00	+
19:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+
20:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+
21:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+
22:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+
23:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+

Figure 6-16 - Stiffness matrix of the structure

6.1.12 Save the structure

- If running the program as an application, select "Save" from the "File" menu to save the structure in binary format on the local hard drive. The saved structure can be retrieved later by selecting "Open" from the "File" menu.

To save the structure in the StructureXML text format, select "View StructureXML" from the "File" menu. This will bring up the StructureXML text window. Next select "Save" from the "File" menu to save the XML text on the local hard drive.

- Running the program both as an applet and as an application let's you save your structure to the server over the Internet. Select "Save to Server" from the "File" menu. This will upload the structure in binary format to the server running the applet. The file can later be retrieved by selecting "Download from Server".

To save the structure in the StructureXML format when running as an applet, select "View StructureXML" from the "File" menu. This will bring up the StructureXML text window. Next, select "Select All", followed by "Copy" from the "Edit" menu and paste the copied text to any text editor and save it on the local hard drive. To retrieve a structure from the XML later, open the XML text using any text editor, copy the StructureXML, and paste it on the

text window from the “*New StructureXML*” option in the “*File*” menu and hit the “*OK*” button.

6.2 Functions available

Frameworks allows you to model any type of framed structure. The click and drag interface lets you input two-dimensional structures, but the program can take in and perform calculations for three-dimensional structures as well using a text based XML input (See Section 2.4.2). This section explains the functions available in Frameworks and Trussworks to create structures using the click and drag interface, to save the structure and retrieve them in binary format. It also describes how to perform computations.

The functionalities available can be broken into four main groups, which are also the menu items that are present in the program. The first thing that is seen on starting the programs is the dialog box in Figure 6-17. This allows you to pick the units that you will be using for your structure, as well as the initial screen dimensions and the grid spacing. The grid spacing is also the snap resolution used in the program. Both the screen dimensions and the grid spacing can be changed later at any time using the *Zoom* and *Set Grid Spacing* functions (Section 6.2.2). Once the units have been set, the only way to change them for the same structure is by editing the StructureXML text (Section 2.4.2).¹⁰

¹⁰ A menu item that lets you change units after the initial selection is made is included in the program, but was “turned off” from the final version to reduce clutter in the menu bar. To turn the functionality back on, simply download the source code, “uncomment” the part that adds the menu-item to the *Function* menu, and compile the programs using the Java SDK (See Section 1.1).

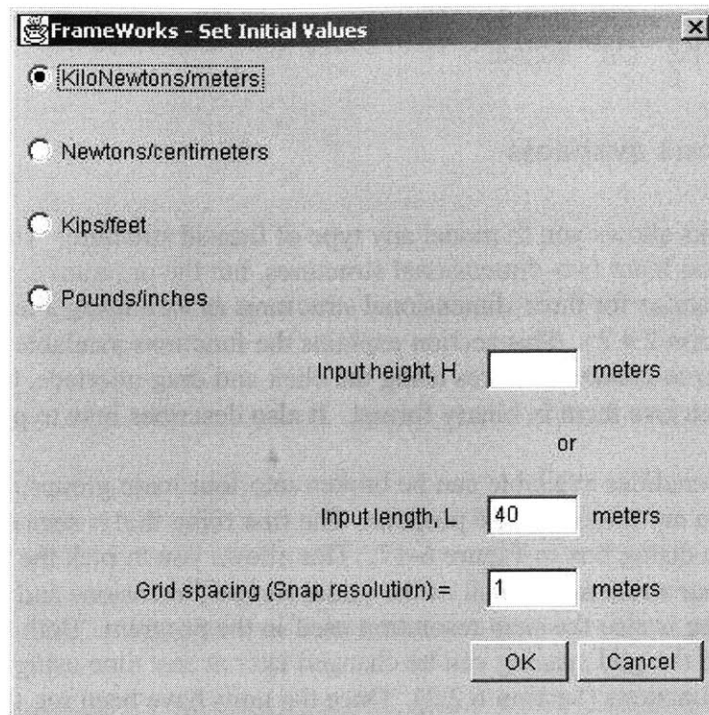


Figure 6-17 - Start Options

6.2.1 File

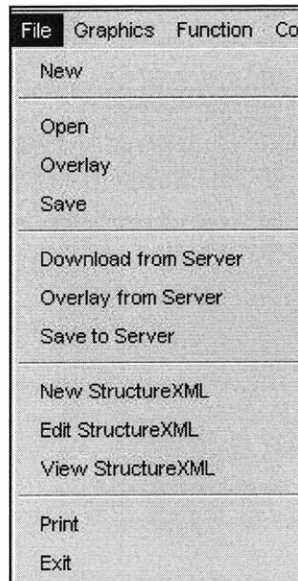


Figure 6-18 - File Menu Functions

- **New** – Creates a new structure and clears the old structure from memory.
- **Open** – Opens a structure saved in binary format from the users hard drive. *Not available when running as an applet.*

- **Overlay** – Overlays another structure saved in binary format from the users hard drive onto the existing structure. The program increments the node numbers of the second structure with the number of nodes present in the first structure. *Not available when running as an applet.*
This would allow, for example, different users to work on different parts of a structure. Once the parts have been created and saved, they can all be overlaid to form the whole structure.
- **Save** – Saves structure in binary format on users hard drive. *Not available when running as an applet.*
- **Download from Server** – Downloads structure saved in binary format in a server over the Internet using TCP¹¹. *Applets can only download structures from a server that has the same host name as the web server from which it originated.*
- **Overlay from Server** – Overlays another structure saved in binary format by downloading the second structure from a server over the Internet using TCP. *Applets can only overlay structures from a server that has the same host name as the web server from which it originated.*
- **Save to Server** – Saves structure in binary format to a server over the Internet using TCP. *Applets can only save structures to a server that has the same host name as the web server from which it originated.*
- **New StructureXML** – Starts a new StructureXML document (see Section 2.4.2). If running the programs as an application, then File → Open opens a dialog box that lets the user open a text (XML) document from their hard drive. *Users running the program as an applet can copy a StructureXML text from a local file, and paste it on the textarea.*
- **Edit StructureXML** – Lets the user edit their structure as a text (XML) file. The user must press the *Apply* button to apply changes.
- **View StructureXML** – Lets the user view their structure in the StructureXML format. If running as an application, File → Save lets the user save the structureXML document in their hard drive. *Applet users who want to save the StructureXML text must highlight all the text in the textarea, copy it, and then paste it on a text editor to save it on their hard drive.*
- **Exit** – Closes the program and clears the structure from memory.

¹¹ TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers [Campione 00].

6.2.2 Graphics

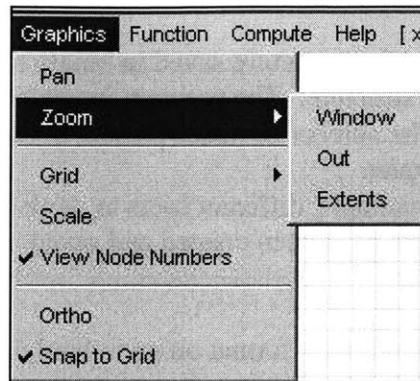


Figure 6-19 – Graphics Menu Functions

- **Pan** – Allows the user to “pan” or move a structure around the screen by clicking and dragging the mouse.
- **Zoom**
 - **Window** – Zooms *in* on a structure, in the rectangle created by clicking and dragging the mouse on the screen.
 - **Out** – Zooms *out* by a factor of 0.5x.
 - **Extents** – Zooms to fill the full screen with the structure.
- **Grid**
 - **On** – Lets the user turn on, or turn off the grid lines.
 - **Set Grid Spacing** – Allows user to explicitly set the grid spacing. This is also the “snap to grid” resolution used by the program.
- **Scale** – Toggles the scale visibility to on or off.
- **View Node Numbers** – Toggles node numbering to on or off.
- **Ortho** – Toggles ortho mode to on or off. When in ortho (orthogonal) mode, the user will only be able to draw horizontal and vertical members.
- **Snap to Grid** – Toggles snap to grid mode to on or off. When in snap to grid mode, the start and end of members created by clicking the mouse will snap to the grid lines defined above.

6.2.3 Function

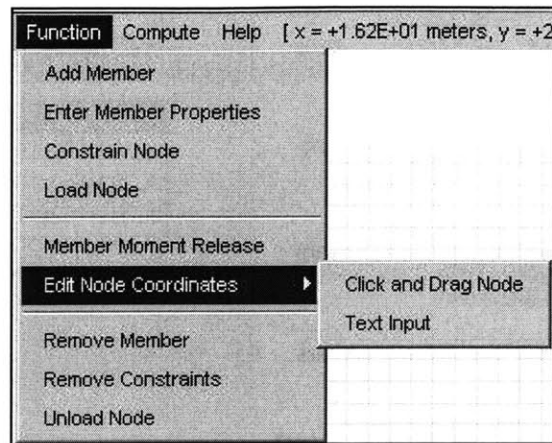


Figure 6-20 - Function Menu Functions

- **Add Member** – Adds a member by clicking the mouse twice. The first click records the first Node coordinates, and the second click records the second Node coordinates.
- **Enter Member Properties** – Brings up the Edit Member Properties window¹², which allows the user to enter the member properties for members. The user can set values for the member properties for all the members at once by selecting “*Apply to all members*” (Figure 6-21, and Figure 6-22), and after setting the member properties, clicking on the “*Apply*” Button. To set values member property values for individual members, first select “*Apply to selected members*” after setting the values. Next go back to the main screen and click on the members you want to select. Selected members appear blue.

Trussworks brings up the window in Figure 6-21 that allows the user to set the values of the cross-sectional areas and the modulus of elasticity. This can be done by either entering the values directly in the textboxes on the left, or by selecting the pre-programmed member cross-sections and materials on the right, which will fill in the values of the textboxes.

¹² This package (see Section 2.3.1) was developed by a colleague, H.P. Nguyen, who is a graduate student at MIT in the department of Civil and Environmental Engineering, and was adapted for use in the programs.

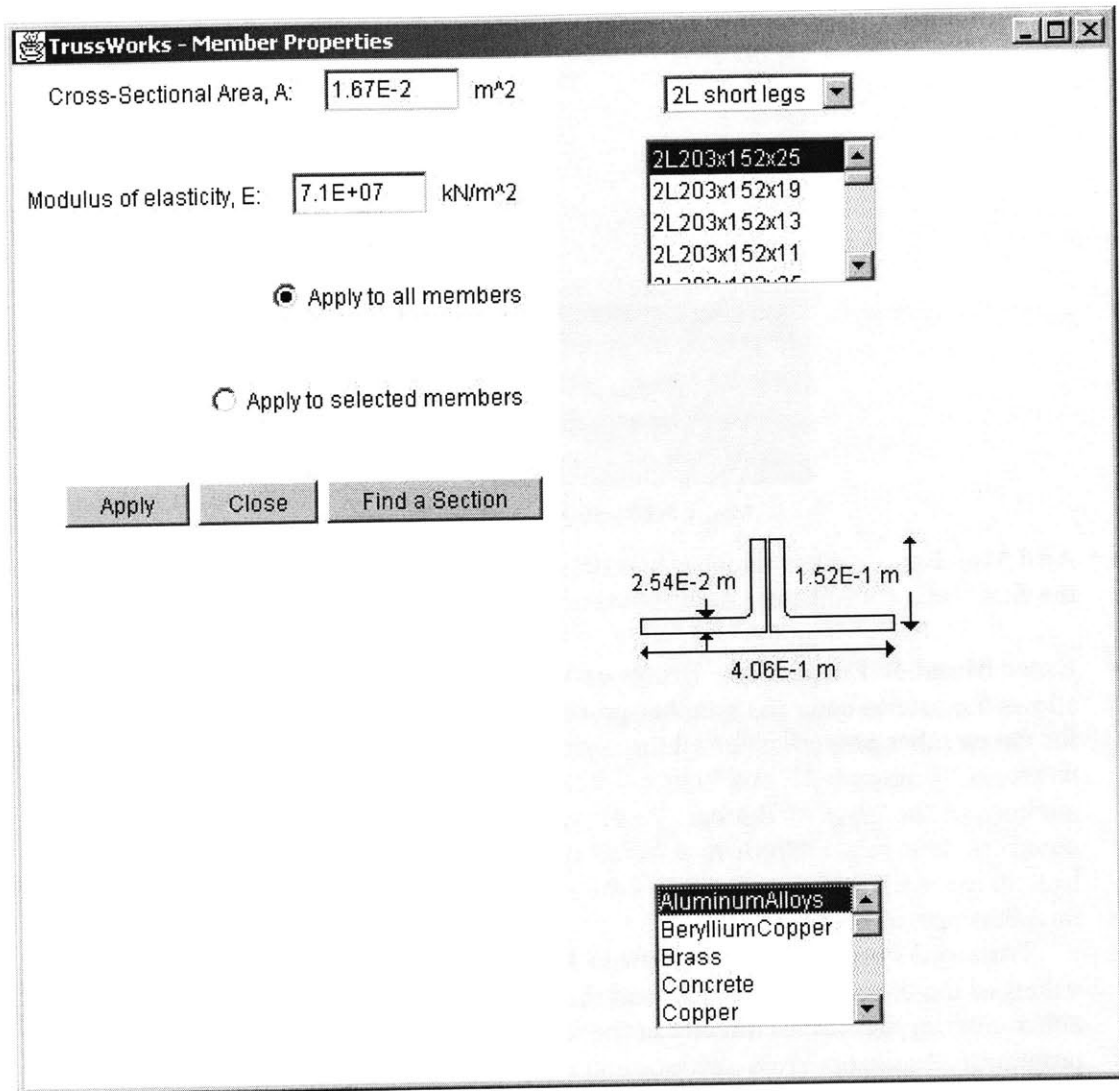


Figure 6-21 - Enter Member Properties Panel (Trussworks)

Frameworks brings up the window in Figure 6-22. In addition to the member cross-sectional area and modulus of elasticity, this window lets the user set values for the Saint-Venant's torsion constant, moment of inertia about the local y and z axis, the shear modulus, and the angle of roll. Here the angle of roll is defined as *"the angle, measured clockwise positive when looking in the negative x direction, through which the local xyz coordinate system must be rotated around its x axis, so that the xy plane becomes vertical with the y axis pointing upward (i.e., in the positive direction of the global Y axis). For the special case of vertical members (i.e., members with centroidal or local x axis parallel to the global Y axis), this angle is defined as the angle, measured clockwise positive when looking in the negative x direction, through which the local xyz coordinate system must be rotated around its x axis, so that the local z axis becomes parallel to, and points in the positive direction of the global Z axis."* [Kassimali 99] The positive x axis, is the axis defined by the start node (the node generated by the first click in the "Add Member" mode) and end node (the node generated by the second click in the "Add Member" mode) of the member.

FrameWorks - Member Properties

Cross-Sectional Area, A: m²

Saint-Venant's torsion constant, J: m⁴

Moment of inertia about local y axis, I_y: m⁴

Moment of inertia about local z axis, I_z: m⁴

Modulus of elasticity, E: kN/m²

Shear Modulus, G: kN/m²

Angle of roll, B: degrees

Apply to all members

Apply to selected members

2L short legs

- 2L203x152x25
- 2L203x152x19
- 2L203x152x13
- 2L203x152x11

2.54E-2 m 1.52E-1 m

4.06E-1 m

- AluminumAlloys
- BerylliumCopper
- Brass
- Concrete
- Copper

Figure 6-22 - Enter Member Properties Panel (FrameWorks)

Clicking on the “*Find a Section*” button brings up the window shown in Figure 6-23. This window allows the user to compare two cross-sections, and also has the functionality to match the section on the right with the section on the left based on the largest section modulus.

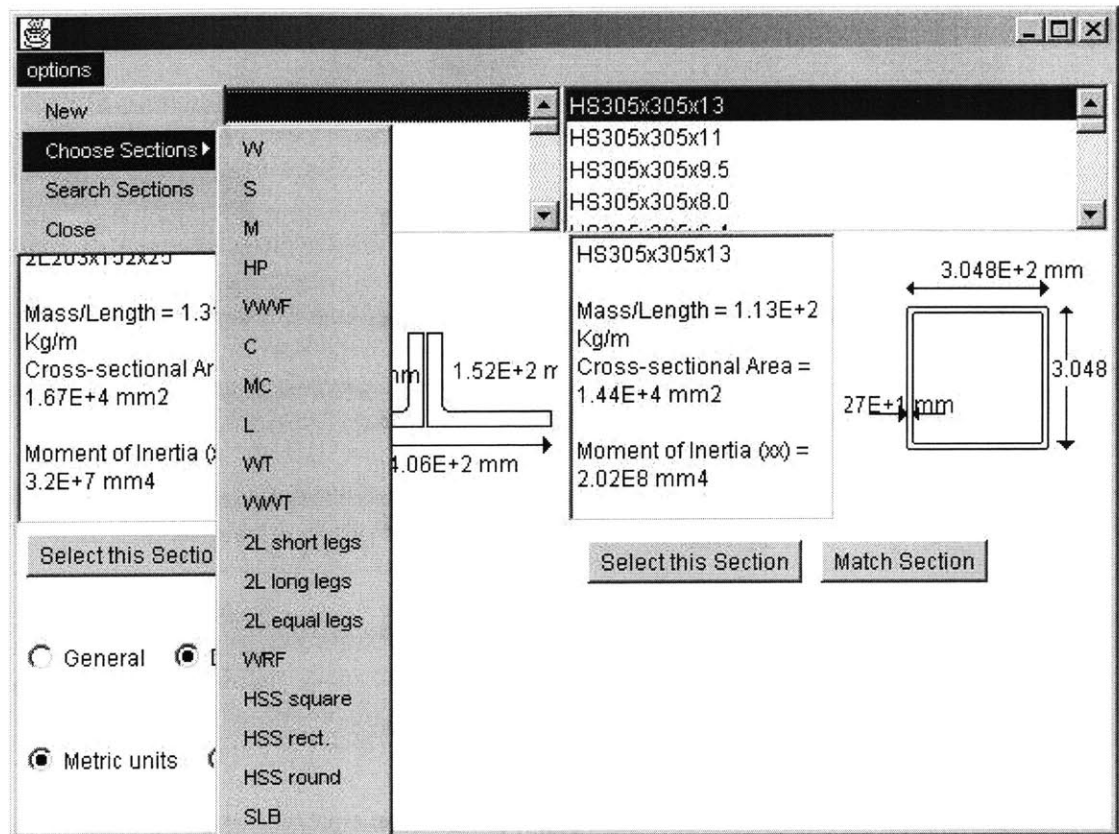


Figure 6-23 - Find Section Panel

Another feature is the “*Search Sections*”, which can be selected from the “Options” menu. This brings up the sub-panel shown in Figure 6-24. This lets you do two things. First, you can search for a section by entering the requirements in the textfields on the bottom for I_x , I_y , M , J , S_x , S_y , and A . If more than one section meets the criteria, they are displayed in the list on top of the panel (Figure 6-24). Second, once a suitable cross-section has been found, the user can click on the “*Select this section*” button. This will automatically select this section, close the window, and show up in the “*Enter Member Properties*” panel (Figure 6-21, and Figure 6-22).

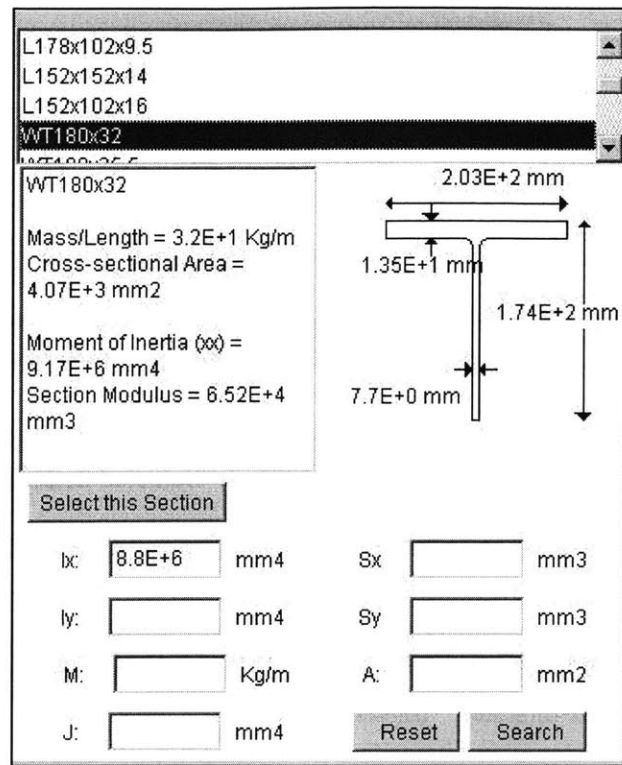


Figure 6-24 - Search Section sub-panel

- **Constrain Node** – Allows the user to constrain nodes. Trussworks brings up the window shown in Figure 6-25 and allows the user to constrain the nodal displacements in the x, y, and z direction.

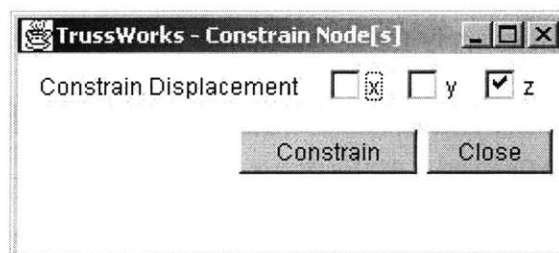


Figure 6-25 - Constrain Node window (Trussworks)

Frameworks brings up the window shown in Figure 6-26, and in addition to constraining the nodal displacements, also lets the user constrain the nodal rotations in the x, y, and z directions.

Once the window comes up, go back to the main screen, and click on nodes to select. Selected nodes have a blue box drawn around them. Finally, click on the "Constrain" button on the "Constrain Node" window.

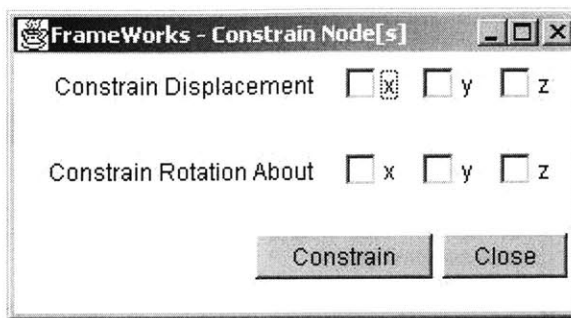


Figure 6-26 - Constrain Node window (Frameworks)

- Load Node** – Allows the user to load nodes. Nodes are selected by going back to the main screen and clicking on the nodes to select. Selected nodes have a blue box drawn around them. Trussworks brings up the window shown in Figure 6-27, which lets the user apply nodal forces in the x, y, and z directions.

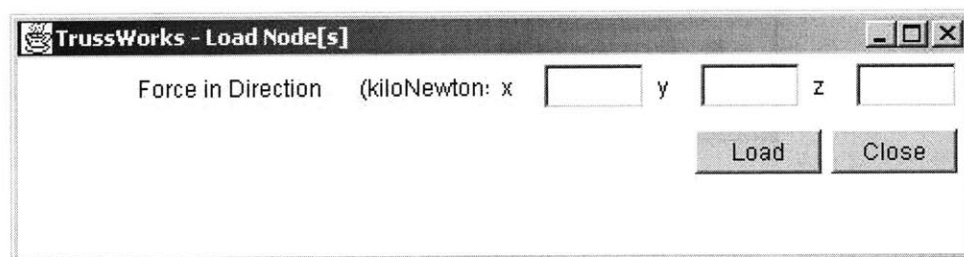


Figure 6-27 - Load Node window (Trussworks)

Frameworks brings up the window shown in Figure 6-28, which in addition to nodal forces, also lets the user apply nodal moments in the x, y, and z direction.

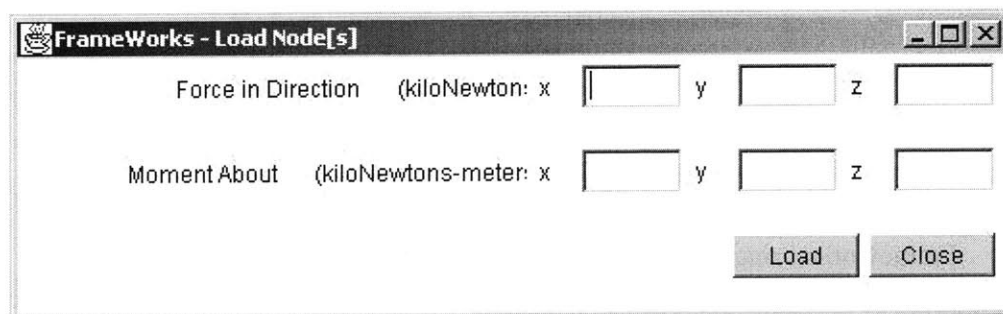


Figure 6-28 - Load Node window (Frameworks)

In both cases, the x, y, and z directions refer to the *global* coordinates of the system.

- Member Moment Release** - Allows the user to release moments on a member (only available in Frameworks). Releasing a moment creates a hinge at the end of the member, making it incapable of transferring moment forces, much like a truss member. This lets you create a frame structure, with truss elements. Select this

option, and then click on a member to release moments. This will bring up the dialog box shown in Figure 6-29. Check the node where the moment is to be released and then click on the “Release” button. Released moments show up as a circle on the main window (Figure 6-30 shows the moment released at node 1).

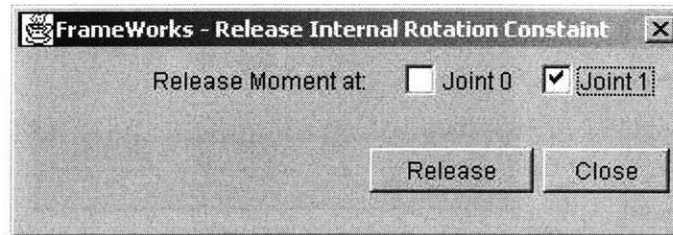


Figure 6-29 - Release Moment Dialog-box

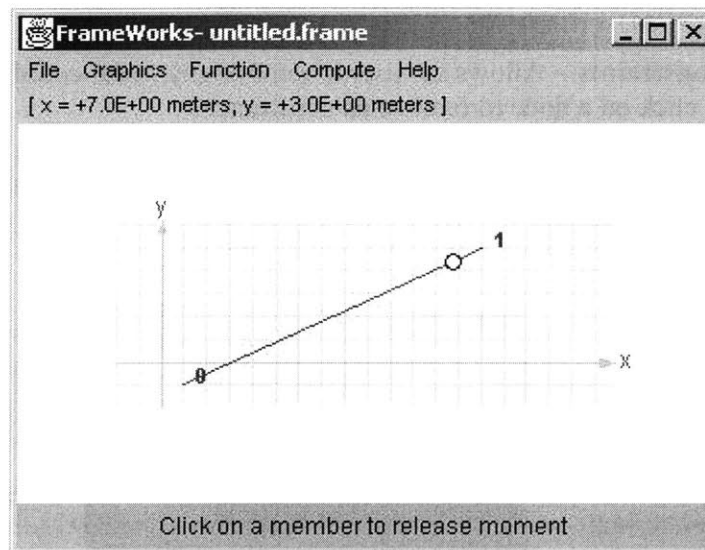


Figure 6-30 - Member moment released at node 1

- **Edit Node Coordinates** – Allows the user to change node coordinates. This can be done in two ways:¹³
 - **Click and Drag Node** – This option lets the user change a node coordinate by clicking on a node to select it, and then clicking again to set the new coordinates.
 - **Text Input** – This option lets the user change the node coordinates by typing the new coordinates. First select this option, and then click on any node. This will bring up the dialog box shown in Figure 6-31. Simply enter the new coordinates in the textboxes labeled x, y, and z and hit the “OK” button.

¹³ An additional way to edit node coordinates is by editing the StructureXML text (Section 2.4.2). This can be done by selecting the *File* → *Edit StructureXML* menu-item.

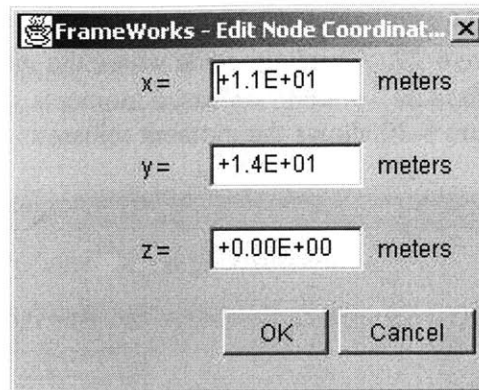


Figure 6-31 - Edit Node Coordinates - Text Input

- **Remove Member** – Allows the user to remove members from your structure. Select this option, and then click on a member to remove.
- **Remove Constraints** – Allows the user to remove any nodal coordinates. Select this option, and click on a node to remove its constraints.
- **Unload Node** – Allows the user to unload a node. Select this option, and click on a node to unload.

6.2.4 Compute

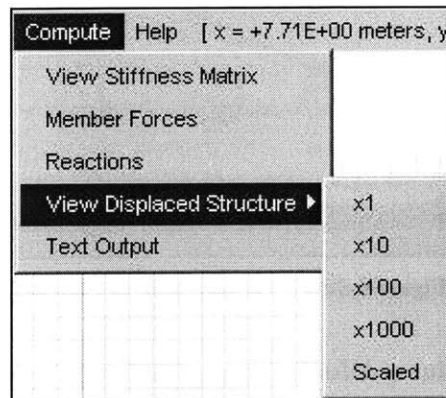


Figure 6-32 - Compute Menu Functions

- **View Stiffness Matrix** – This option brings up a window that displays the stiffness matrix (Section 1.1) of the structure as shown in Figure 6-33.

The screenshot shows a window titled "FrameWorks - Stiffness Matrix" with a menu bar containing "File" and "Edit". The main area displays a 21x6 matrix of values. The first two columns contain non-zero values, while the remaining four columns are mostly zeros, with some non-zero values in the first few rows. The values are in scientific notation.

Row	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
1:	+8.62E+04	+1.34E+05	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
2:	+1.34E+05	+2.17E+05	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
3:	+0.00E+00	+0.00E+00	+6.14E+01	+2.45E+02	-1.53E+02	-8.07E+02
4:	+0.00E+00	+0.00E+00	+2.45E+02	+1.31E+03	-8.07E+02	+5.80E+03
5:	+0.00E+00	+0.00E+00	-1.53E+02	-8.07E+02	+5.80E+03	+0.00E+00
6:	-9.28E+03	+5.80E+03	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
7:	-8.62E+04	-1.34E+05	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
8:	-1.34E+05	-2.17E+05	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
9:	+0.00E+00	+0.00E+00	-6.14E+01	-2.45E+02	+1.53E+02	+8.07E+02
10:	+0.00E+00	+0.00E+00	+2.45E+02	+6.48E+02	-4.22E+02	+2.45E+02
11:	+0.00E+00	+0.00E+00	-1.53E+02	-4.22E+02	+2.45E+02	+0.00E+00
12:	-9.28E+03	+5.80E+03	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
13:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
14:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
15:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
16:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
17:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
18:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
19:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
20:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00
21:	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00	+0.00E+00

Figure 6-33 - Stiffness matrix for a sample frame structure

- Member Forces** – Computes the internal member forces that exist on a structure. Select this option, and then click on any member to view its internal forces. This brings up a dialog box as shown in Figure 6-34. (Since trusses can only carry axial loads, selecting this option only displays the axial member forces in Trussworks).

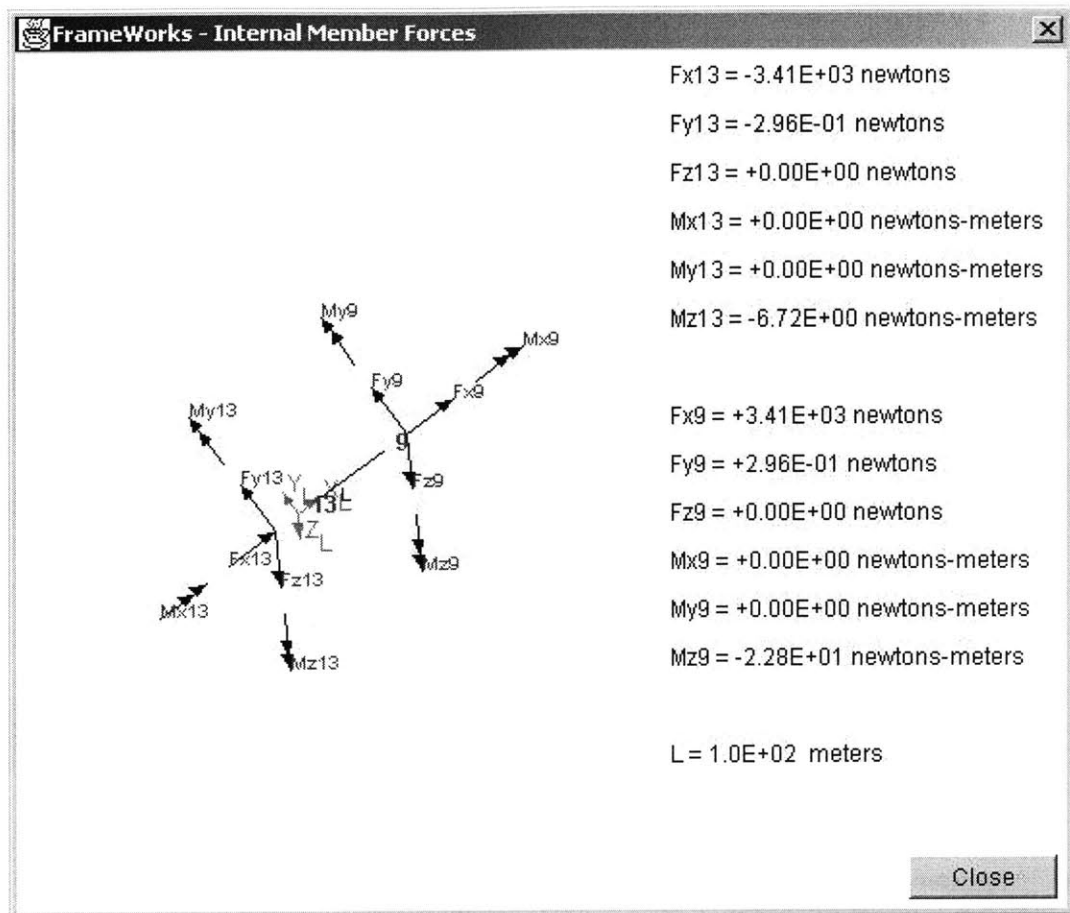


Figure 6-34 - Dialog box showing member forces

- **Reactions** – Computes and displays the reactions on the screen for the constrained nodes. Figure 6-35 shows the reactions for a sample truss.

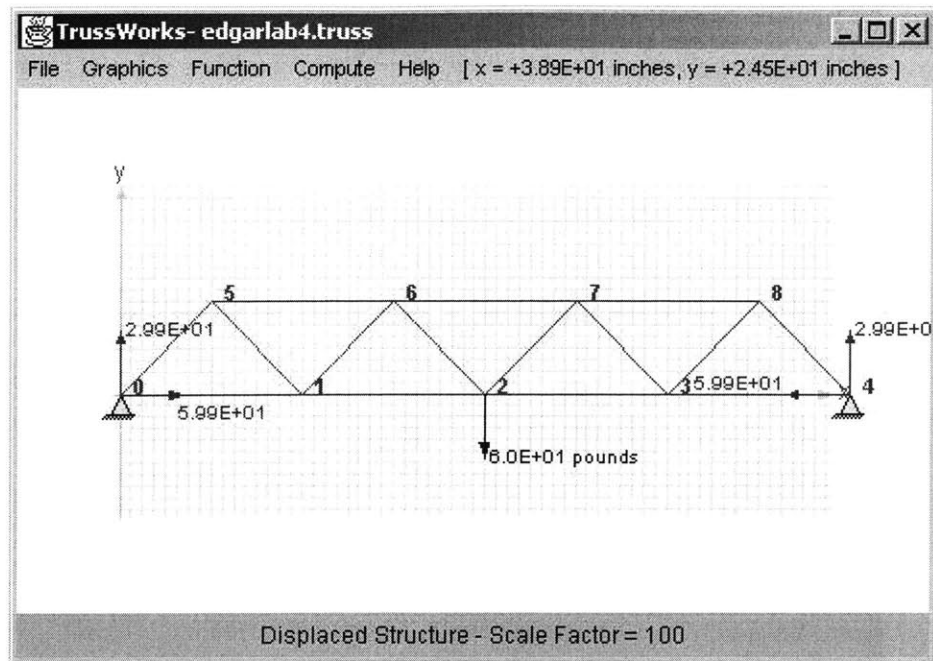


Figure 6-35 - Compute Reactions

- **View Displaced Structure** – Computes the nodal displacements, and shows the displaced structure on the screen.
 - **X1** – Shows exact displacements, i.e. Scale factor = 1.
 - **X10** – Magnifies displacements by a factor of 10.
 - **X100** – Magnifies displacements by a factor of 100.
 - **X1000** – Magnifies displacements by a factor of 1000.
 - **Scaled** – Magnifies displacements by an appropriate scale to properly view displacements.

Figure 6-36 shows a sample displaced structure, with a displacement magnification factor of 100.

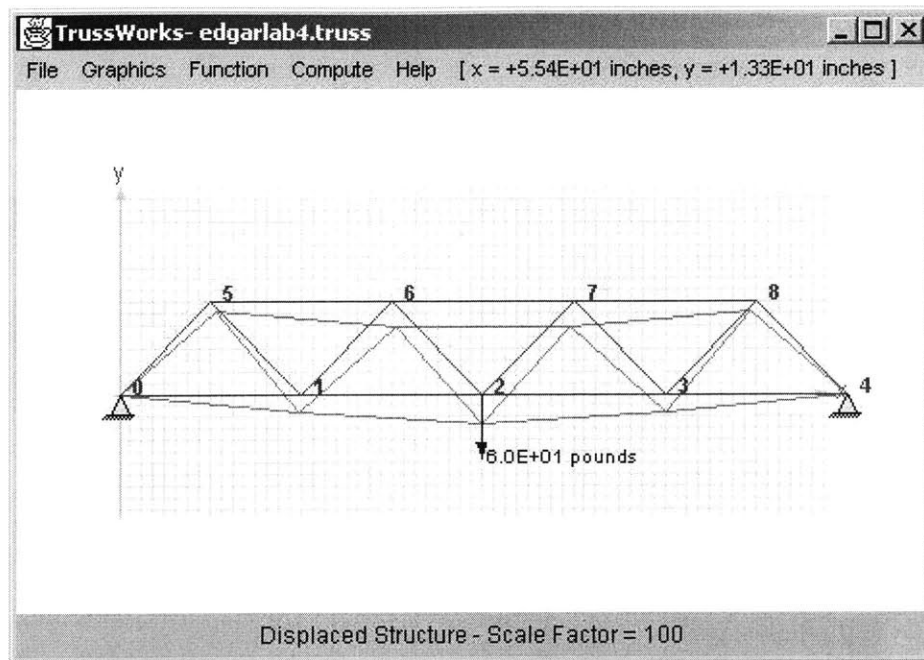


Figure 6-36 - View Displaced structure (Scale Factor = 100)

- **Text Output** – Opens another window that lists the reactions, member forces, and displacements of the structure as shown in?

The screenshot shows a window titled "FrameWorks - Text Results" with a menu bar containing "File" and "Edit". The main content area displays the following text output:

```

Node 16:      -2.49E+03   +1.38E+04   +0.00E+00
Node 17:      -4.25E-02   +3.78E-01   +0.00E+00
Node 18:      -2.93E-02   -3.43E-01   +0.00E+00
Node 19:      -9.03E-03   +1.12E+00   +0.00E+00
-----
Reaction Moments - newtons-meters
                Mx:      My:      Mz:
Node  0:      +0.00E+00   +0.00E+00   +6.26E+00
Node 16:      +0.00E+00   +0.00E+00   +8.40E+00
-----
Member Forces - newtons
                Fx:      Fy:      Fz:
Member 1 - 2: (Node 1) -2.21E+03   +8.28E-02   +0.00E+00
Member 1 - 2: (Node 2) +2.21E+03   -8.28E-02   +0.00E+00
Member 2 - 3: (Node 2) -4.99E+03   -4.06E-01   +0.00E+00
Member 2 - 3: (Node 3) +4.99E+03   +4.06E-01   +0.00E+00
Member 3 - 4: (Node 3) -4.99E+03   -2.41E-01   +0.00E+00

```

Figure 6-37 - Text output for a sample frame structure

6.3 The Stiffness Matrix

Simply put, the stiffness matrix of a structure, K_{ij} is the reaction produced at the i^{th} degree of freedom due to a unit displacement at the j^{th} degree of freedom. If a structure is prescribed a unit displacement at one degree of freedom, and constrained at all the possible degrees of freedom, then the reactions produced at those constrained nodes would make up the column of the stiffness matrix corresponding to the selected degree of freedom. The stiffness matrix is diagonally dominant and symmetric.

Once the stiffness matrix has been formulated, the unknown displacements can be calculated using the following relation:

$$[K][u] = [P] \quad \text{Equation 6-1}$$

Where u is the displacement vector, and P is the load vector.

To illustrate, consider a two-dimensional truss structure (

Figure 6-38). The u 's are the possible motions of the two nodes along the member's local x and y axis (any other motion of the nodes can be formulated as a combination of these motions).

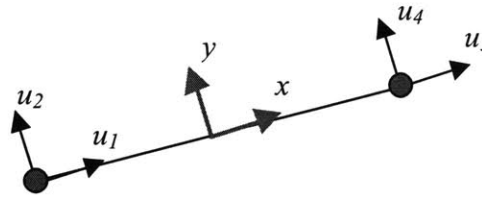


Figure 6-38 – local axis of member and the degrees of freedom

Since the truss member has four degrees of freedom, its stiffness matrix is a 4x4 matrix in the following form:

$$\underline{K} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix}$$

To generate the above stiffness matrix, \underline{K} , first constrain the two nodes as shown in Figure 6-39.

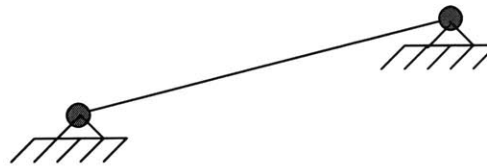


Figure 6-39 – Constrain Nodes

Next, apply a unit displacement in the direction of the first degree of freedom, u_1 (Figure 6-40). The reactions created at the nodes make up the first column of the \underline{K} matrix, which can be computed using Hooke's Law which states that in the elastic range of a material, $\delta = \frac{PL}{EA}$, where δ is the amount the member deforms when a force P is applied to it. (L is the original length of the member, A is the cross-sectional area, and E is the modulus of elasticity). In our case, P is the force generated, or the reaction at the nodes, when the member is deformed by an amount δ , which in our case is one.

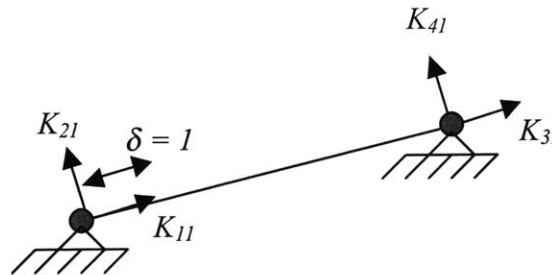


Figure 6-40 – Apply a unit displacement in the direction of u_1

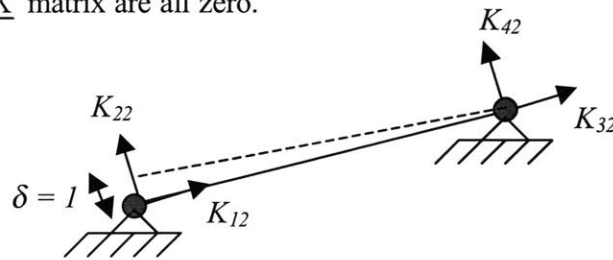
Assuming the force to be positive along the positive local axis of the member (Figure 6-38), and since the deformation, $\delta = 1$, then $K_{11} = P = \frac{\delta EA}{L} = \frac{EA}{L}$. Summing the forces in the x direction then gives K_{31} . That is, $\sum F_x = 0 = K_{11} + K_{41}$. Therefore, $K_{41} = -\frac{EA}{L}$. The vertical reactions, K_{21} and K_{31} , are of course both zero. This then gives us the first column of the \underline{K} matrix –

$$\begin{bmatrix} k_{11} \\ k_{21} \\ k_{31} \\ k_{41} \end{bmatrix} = \begin{bmatrix} \frac{EA}{L} \\ 0 \\ -EA \\ 0 \end{bmatrix}$$

Applying a unit displacement in the direction of the third degree of freedom, u_3 , similarly gives us the third column of the \underline{K} matrix.

$$\begin{bmatrix} k_{13} \\ k_{23} \\ k_{33} \\ k_{43} \end{bmatrix} = \begin{bmatrix} -\frac{EA}{L} \\ 0 \\ EA \\ 0 \end{bmatrix}$$

Finally, apply a unit displacement in the direction of the second and fourth degrees of freedom, u_2 and u_4 . Figure 6-41 shows a unit displacement in the second degree of freedom. Assuming that these vertical displacements are large in comparison to the length of the member, there are no reactions at the nodes, and therefore the second and fourth columns of the \underline{K} matrix are all zero.

**Figure 6-41 - Apply a unit displacement in the direction of u_2**

The stiffness matrix for a two-dimensional truss member, \underline{K} , then takes on the following form –

$$\underline{K} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} \\ k_{21} & k_{22} & k_{23} & k_{24} \\ k_{31} & k_{32} & k_{33} & k_{34} \\ k_{41} & k_{42} & k_{43} & k_{44} \end{bmatrix} = \begin{bmatrix} \frac{EA}{L} & 0 & -\frac{EA}{L} & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & \frac{EA}{L} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \frac{EA}{L} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Once the stiffness matrices for each truss member in a structure has been formulated, the stiffness matrix for the entire structure can then be created by matching the degree of freedom of the individual members to the degree of freedom of the structure. This method can be carried out for three-dimensional truss structures as well, and can also be extended to include two and three-dimensional framed structures.

The stiffness matrix can then be used in Equation 6-1, and then if the loading conditions are known, then the displacements can be calculated by solving the simultaneous equations. The member forces, as well as the reactions can also computed using the stiffness matrix.¹⁴

6.4 System requirements to run program

6.4.1 As an Applet

To run the programs as an applet, the browser must have the Java virtual machine. The programs do not use swing and run on Java version 1.1 and any newer version. Java is a cross platform language, and as long as the virtual machine exists, it can be run on any browser. Frameworks and Trussworks place no limits on the number of members that can be added to the structure, but the RAM may impose a limit, as the number of members gets significantly larger.

6.4.2 As an Application

To run the program as an application, the local machine must have the Java virtual machine that can be downloaded from the Sun's web page. The earliest version of Java, which will work with Frameworks and Trussworks, is version 1.1. Similarly to running the programs as an applet, Frameworks and Trussworks place no limits on the number of members that can be added to the structure, but the RAM may impose a limit, as the number of members gets significantly larger.

¹⁴ The *flexibility matrix* of a structure, F_{ij} , on the other hand, is deflection produced at the i^{th} degree of freedom due to a unit force applied at the j^{th} degree of freedom. The stiffness matrix is the inverse of the flexibility matrix, and it is sometimes more convenient to formulate the stiffness matrix in this manner. Note: You can formulate a stiffness matrix for an unrestrained structure. You cannot do that for the flexibility matrix.