An object oriented design for finite element analysis
by Terrance E Kubat

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University

Abstract:
Finite element methods are used to solve a variety of problems in engineering. They have evolved into sophisticated numerical analysis techniques requiring powerful computer system solutions. Programs implementing these methods tend to be complex both in development and maintenance. With significant research and new developments in finite element technology, code evolution is a significant issue.

Finite element software is generally written by engineers using a procedural language like FORTRAN. This results in additional complexity to the system: the engineer's model of the problem must be transformed to meet the requirements of the programming language. Unfortunately the code obscures many of the concepts used in the finite element method.

It becomes difficult to see what the program is modelling, making system verification and modification that much harder. A programming system which captures the higher level concepts of the method and allows for this evolution is desired.

This thesis builds upon object oriented principles of design to create an extensible system for building finite element analysis programs. The design has been adapted from previous work done in the Common Lisp Object System. The C++ programming language is chosen as a more appropriate vehicle for implementing this engineering tool, which is a set of high-level finite element data types. Thede abstract data types define the objects in the engineer's model and therefore bring the software closer to the terminology used in finite element methods.

The result of this project is a base library for finite element analysis. This library provides a foundation upon which finite element applications can be built. An example program demonstrates the library for two dimensional structural trusses. This application is described and followed by a discussion of future library expansion.

AN OBJECT ORIENTED DESIGN FOR FINITE ELEMENT ANALYSIS

by

Terrance E. Kubat

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 1992

APPROVAL

of a thesis submitted by

Terrance E. Kubat

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

11/25/92
Date

Ray S. Babcock
Chairperson, Graduate Committee

Approved for the Major Department

11/25/92
Date

J Dunbigh Stanley
Head, Major Department

Approved for the College of Graduate Studies

12/14/92
Date

R L Brown
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission of extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signed Thomas E. Talbot

Date 25 NOV '92

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

LIST OF FIGURES

ABSTRACT

Finite element methods are used to solve a variety of problems in engineering. They have evolved into sophisticated numerical analysis techniques requiring powerful computer system solutions. Programs implementing these methods tend to be complex both in development and maintenance. With significant research and new developments in finite element technology, code evolution is a significant issue.

Finite element software is generally written by engineers using a procedural language like FORTRAN. This results in additional complexity to the system: the engineer's model of the problem must be transformed to meet the requirements of the programming language. Unfortunately the code obscures many of the concepts used in the finite element method. It becomes difficult to see what the program is modelling, making system verification and modification that much harder. A programming system which captures the higher level concepts of the method and allows for this evolution is desired.

This thesis builds upon object oriented principles of design to create an extensible system for building finite element analysis programs. The design has been adapted from previous work done in the Common Lisp Object System. The C++ programming language is chosen as a more appropriate vehicle for implementing this engineering tool, which is a set of high-level finite element data types. These abstract data types define the objects in the engineer's model and therefore bring the software closer to the terminology used in finite element methods.

The result of this project is a base library for finite element analysis. This library provides a foundation upon which finite element applications can be built. An example program demonstrates the library for two dimensional structural trusses. This application is described and followed by a discussion of future library expansion.

CHAPTER 1

INTRODUCTION

Problem

As a numerical method, finite element analysis has been developing alongside the digital computers which enabled its practical usage. The finite element method has become a premier technique employed to solve a wide variety of engineering problems. Computer implementations of the finite element analysis (FEA) tend to be very large, complicated FORTRAN programs which obscure the mathematical and engineering concepts which define the method. This creates a problem for those who must maintain existing programs, as well as for developers wishing to expand capabilities to reflect recent advances in FEA technology.

The problem here is twofold. First, FEA ideas are obscured in a programming language which forces a transformation from the application domain into the digital machine's capabilities [Abelson 85]. FORTRAN was invented to solve this very problem, that is, to raise the level of abstraction in programming. In its time, and over the years it has proven to be a great success: allowing scientific formulas to be written out in a program which is more readable than the equivalent machine (or assembly) language program. It does not go far enough however, failing to capture higher level concepts required for today's complex applications.

Secondly, the organization of a typical FEA implementation is rigidly bound to the types of elements used and problems to be solved. Specifically, a change in the implementation of a data structure likely results in rewriting all modules which access that structure. As finite element analysis is still a very active area of research, this 'hard

coding' of modules and data types severely limits the extension of a program to provide advanced capabilities utilizing newly invented technologies. At the same time, developing a new implementation from scratch can be quite expensive in both time and money [Nagy 78].

Goals

The purpose of this thesis is to investigate a possible solution to these two problems. Ideally, a FEA computer program should read like a textbook on the method. Concepts, terminology, organization, and solution steps should all be preserved in the program's modules--at least on the surface, or in the interface. A programming language built upon the concepts of abstraction and inheritance is needed. Obviously, FORTRAN will have to be abandoned in favor of a language which will better support programming in the domain of the application. This creates somewhat of a dilemma however. Engineers are the developers, maintainers, and users of FEA programs and typically have only received training in procedural language programming: generally FORTRAN. While the best languages for raising the abstraction level of a program fall into the object oriented category, they generally require a very different paradigm.

Fortunately there are a number of hybrid object oriented languages available today which bridge this gap. The one chosen for this thesis is C++ [Cox 86] [Wiener 88]. Due to its very popular 'parent' language C, which has become somewhat of a universal procedural language, C++ has become a very popular language in recent years. Popularity aside however, C++ provides a solid base upon which to build a numerically efficient, yet readable, extendable library for finite element analysis.

Scope

It is intended here, that a prototype system of finite element data types be created and demonstrated. This system is meant to investigate, and demonstrate a base upon which a complete FEA implementation may later be developed. An exemplar program has been written to allow some experimentation with the ideas. No attempt has been made in addressing the issues of a user interface, pre- or post-processing. Similarly, flow of control issues resulting from parallelism or event-driven environments are not considered. The focus has been made on a representation of finite element concepts within the internal structure of the library.

CHAPTER 2

DESIGN OF THE FINITE ELEMENT LIBRARY

Introduction to the Finite Element Method

Numerical Analysis Technique

The finite element method is a computer oriented analysis technique used by engineers in a wide variety of application areas. It has been applied to problems ranging from heat conduction to fluid flow to structural analysis [Clough 89]. The method involves approximating the behavior of a continuous medium with an imaginary mesh of simple elements. Each element is defined by a small set of interconnection points called nodes. Nodes are defined to represent the boundary of the continuum and also represent arbitrary points within it. By choosing elements and their behaviors carefully, an approximation can be made to the behavior of the continuum. For the remainder of this chapter, a focus will be made on the displacement formulation of finite element analysis as used for a static, linear analysis of structures made of elastic materials.

Mathematical Modelling

In representing a mathematical model for structural analysis several items need to be addressed, including the geometry, the material properties, the boundary (or support) conditions, and the applied loads and forces. While in many cases 'exact' elasticity solutions have been formulated for various classes of structural problems, there still exist an infinity of problems for which no known solution exists. It is here where, finite element analysis allows practical numerical approximations to the solution.

To illustrate, consider the bridge shown in the figure 1. A truck, representing a load on the bridge is also shown.

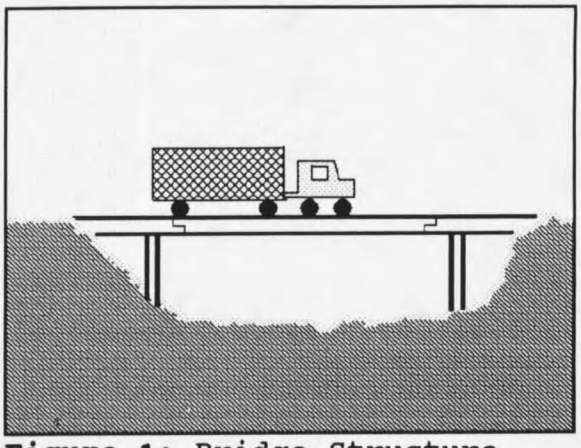

Figure 1: Bridge Structure

As the bridge is loaded, and deflects under the load it is desired to determine its behavior and the stresses it must withstand. (For this particular problem analytical as well as empirical results are readily available.) Figure 2 shows how this bridge might be modeled, with 5 beam-type elements and 6 nodes.

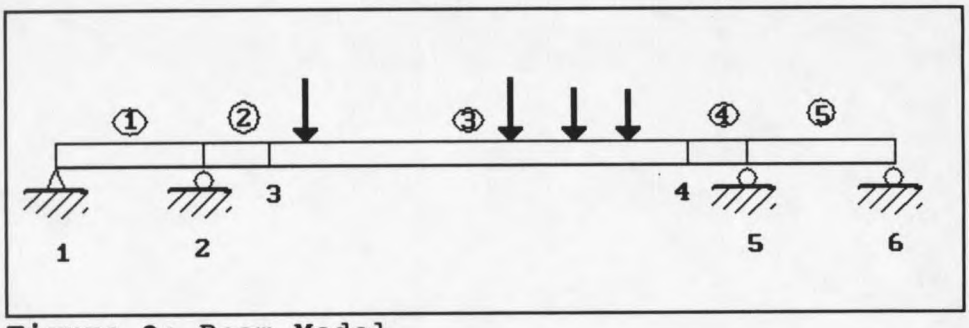

Figure 2: Beam Model

The typical analysis may be made more accurate by using either a larger number of small elements or higher order elements which can better simulate the overall behavior.

The Direct Stiffness Formulation

The method may be summarized in the solution of a system of equations of the following form: [K][d] = [F]. Where [K] is the

'stiffness' matrix of the model, [d] represents the nodal displacements (or degrees of freedom), and [F] is the set of nodal loads on the structure. The system is solved for the unknown, [d]. There are equations of equilibrium for each node in the model, and these are 'assembled' into one set of equations by summing member forces. Once the nodal displacements are found, the results are then fed back into each element to determine stresses within the element.

While the above characterization lends itself to a broad understanding of the method, it leaves out many significant details. As was mentioned earlier, FEA is used on a wide variety of problems; even within structural engineering there are many different classes of problems. The simple one dimensional beam element is just one of many types which have been formulated. Most are more complex, model two or three dimensions, have more than two nodes, or allow for curved boundaries. Some examples are shown in figure 3.

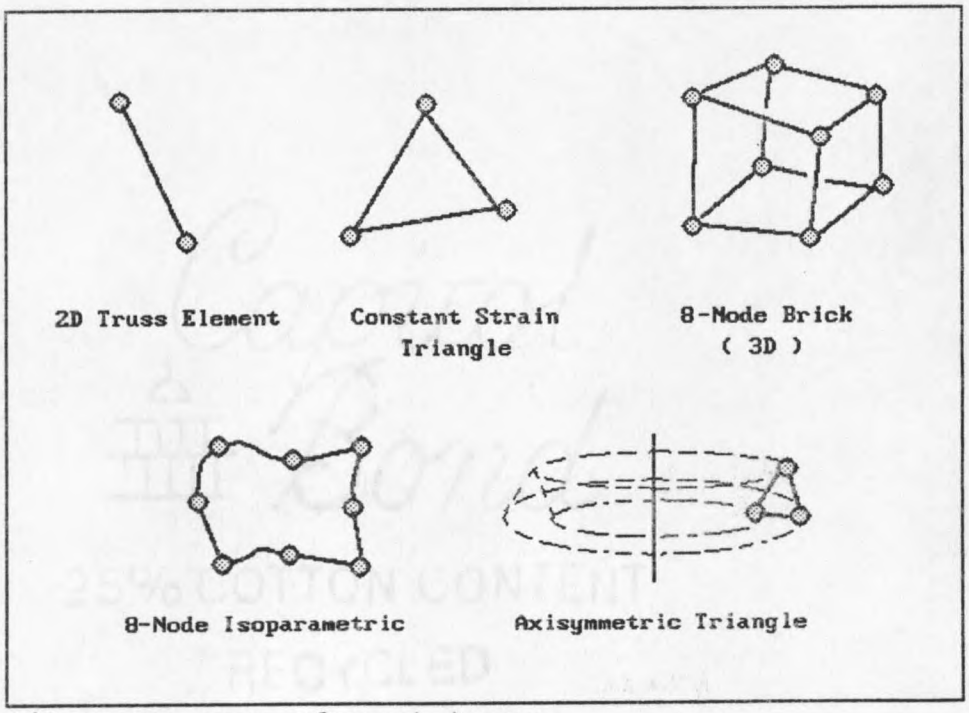

Figure 3: Some element types

Other complications are found in the geometry of the structure, the material used (i.e., steel or wood), and the types of loading. Elements are formulated according to their own 'local' coordinate system, which may in fact be curvilinear. Thus when, assembling the system of equations, or back-solving for element stresses, various coordinate transformations need to take place. Materials are represented mathematically via an elasticity matrix which is used in the calculation of an element's stiffness matrix--a process which requires numerical integration for some element types. Finally, loads which do not coincide with the nodes of the model must be converted into equivalent nodal loads before the system of equations may be solved. For a more rigorous treatment of the method, the reader is referred to the references [Weaver 84] [Zienkiewicz 89].

A Procedural Outline of the Process

General steps for static analysis. [Weaver 84]

1. Create Structure Model
    - A. Read Structural Data
        1. Problem Identification
        2. Material Information
        3. Nodal Information
        4. Element Information
    - B. Assemble Stiffness Matrix
        1. Determine Displacement Indices
        2. Generate the Elasticity Matrix
        3. For each Element
        4. Modify Stiffness for Restraints
2. Create Loading Case
    - A. Enter Loading Data
        1. Nodal Loads
        2. Optional: Other Load Types
    - B. Optional: Compute Equivalent Nodal Loads
        1. Directly or
        2. Using Numerical Integration
        3. Convert to Global Coordinates
    - C. Sum all Load Contributions at each node
3. Solve System of (Nodal Equilibrium) Equations
4. Calculate Results
    - A. Restrained Nodal Reactions
    - B. Element Stresses and Forces

Current Implementations

As mentioned above, current implementations of FEM programs are generally written in FORTRAN, and they are large, difficult to understand and thus hard to modify. Typically, when a new element type

is added to the program, there are a number of subroutines which need to be rewritten. These may be scattered throughout the program. In addition, the storage method for such items as symmetric matrices must be carried explicitly around inside the program. At worse, even the user must be aware of internal data structures as indicated by [Zienkiewicz 89], "The process of specifying the boundary conditions,...is tied to the method adopted to store the global arrays."

John Baugh and Daniel Rehak have done some work in addressing these concerns, and have implemented a system using CLOS (Common Lisp Object System) [Baugh 89A] [Baugh 89B] [Baugh 91]. However, this language, and its functional paradigm, has limited appeal to engineers firmly rooted in procedural based languages. Their ideas and the organization of their library have been used as a pattern for creating the design which follows.

Object Oriented Design

Overview

The object model of design is based on a number of important concepts including abstraction, encapsulation, modularity, and hierarchy [Booch 91] [Fenves 89]. Contrary to many cookbook style approaches to programming, object oriented design is an incremental process by which the application starts with a small and simple form and gradually evolves into the complex system that is required. The breakdown of a problem is facilitated by the natural classifications of objects and ideas in the domain of the application.

Natural Classifications

This is natural because this is how we deal with various layers of complexity in our own environment. For example, when looking at a park, the mind registers 'trees'. When in fact, trees come in all sorts of shapes, sizes and colors. They may be conifers, broad leaf, or even palm trees--a distinction which we choose to make only when necessary.

Each species has its own characteristics which may or may not be important in a given situation. These features are encapsulated within the abstract notion: tree. Similarly, there are often meta-levels in the hierarchy, as a tree is in fact just a plant (while not all plants are trees).

The Process

The process of object oriented design starts with identifying and classifying the domain according to an appropriate level of abstraction. Then, the semantics of the objects must be determined and defined, Finally these objects must be implemented [Booch 91]. This step results in a combined set of data structures and methods (or functions) which operate on the data. Each concept, idea, 'thing' or activity in the domain may be given its own class--a class which is clearly present in the computer program--with its own operations (which are hidden) and an interface (which defines what it is, and how it may be used). Again, this process is repeated as necessary. While uncovering these layers of complexity new features are discovered, perhaps a commonality is recognized, or additional classes are required. The next section shows one possible organization for an FEA library of classes.

An Object Oriented Look at FEA

Identifying Objects and Classes

The design of the finite element library, then, begins with an identification of objects in the domain. Obviously some type of element and node classes will be needed--as these are fundamental to the method. But a closer examination reveals that these are not the most primitive concepts. Indeed, elements and nodes can be quite complex and require a supportive host of classes. A few foundation classes will be needed in order to build the FEA class library.

Foundations

There are two general areas which need support beyond those data types which are built into most modern programming languages. The concepts of vectors and matrices are fundamental mathematical objects of FEA. These are very common tools and therefore much has been done by way of creating classes to capture vector manipulation and algebra. For this thesis project, an early decision was made to take advantage of an existing commercial library--rather than to re-invent similar classes. Rogue Wave's Math.h++ library was employed, with classes representing vectors, matrices, and linear algebraic routines.

The next foundation area is that of geometry. The model representing a true structure must be defined mathematically. Directions and locations of supports and loads, sizes and orientations of elements, as well as the positions of all nodes must be recorded. A simple two dimensional cartesian coordinate class was created for the exemplar system, as well as a coordinate system class which provided for transformations of points between different systems. See figure 4.

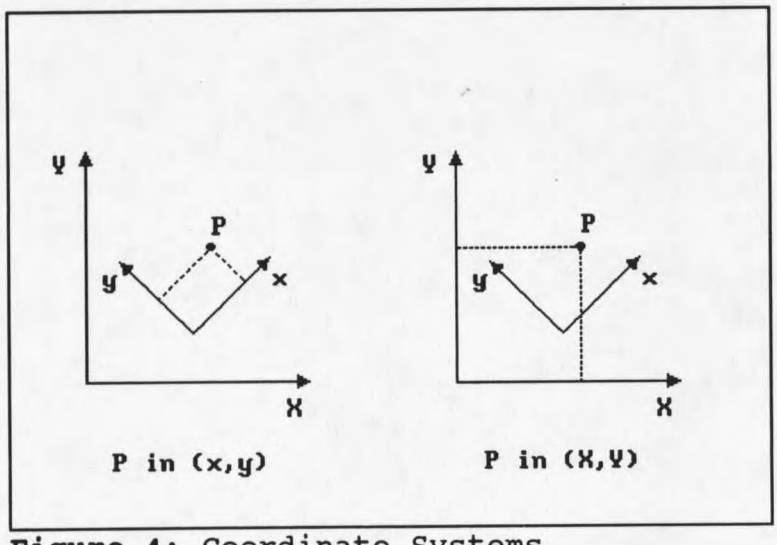

Figure 4: Coordinate Systems

Base Classes

With the foundation laid, an endeavor to capture the fundamental notions of the FEA method is made. This is accomplished with four classes. All of these are related, and the description and definition of each depends to some degree on the others. Perhaps the most independent, is the notion of a material model.

Materials. In the study of elasticity, a material's behavior is defined according to a set of constitutive equations. In the case of three dimensions, these equations relate the six independent stresses to the six corresponding strains. In two dimensions only three equations are required, along with a characterization of the problem type into plane strain or plane stress. This is the general form and can be shared by all specific types of materials--it is however too abstract. For any real material, more information is needed to describe fully these equations. This information comes from empirical values which are readily available for any given material. Furthermore, there are various classes of materials: isotropic, orthotropic, and anisotropic. Each of these, respectively, represents a more complicated behavior and requires more parameters to define. Each of these material types was assigned its own class, which inherits all the properties of the material base class yet exhibits its own specialized behavior.

Degrees of Freedom. Another basic concept embodied within the finite element method is the notion of a degree of freedom. A degree of freedom (DOF) represents the unknown variables at a node. A given node may have many unknowns. In the example given previously of the bridge model, each beam type element was shown with two nodes. What was not shown however, was that each node possessed three degrees of freedom. This is necessary in order to define the possible behavior of a simple beam. The figure below shows a truss element with its four degrees of

freedom, and a possible deflected shape reflecting deflections in two of these directions.

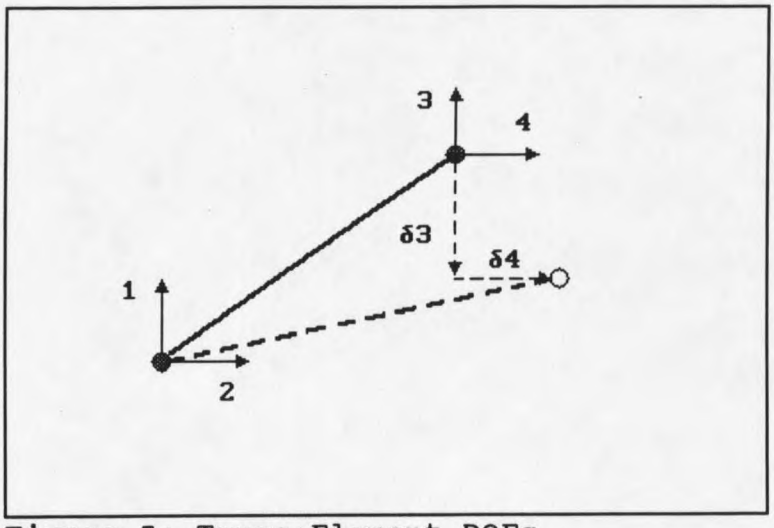

Figure 5: Truss Element DOFs

Degrees of freedom, may be free or supported. A structure model must be adequately supported to prevent its movement in space. A free DOF represents an unknown displacement value (in [D]) which is solved for in [K][D]=[F]. While a supported (or fixed) DOF means that the displacement is zero (or possibly another prescribed value) but the support reaction is unknown. The state of a DOF becomes very important to the correct assemblage of stiffness equations.

It is convenient to record loading information in the DOF as well; it represents an economy of space: because reactions or displacements are known [Zienkiewicz 89]. It is also where the load must be applied. So it is seen that a given node will possess one or more degrees of freedom which may be supported or have loads applied.

Nodes. A node represents a control point in the structural model. This point may be on a boundary, or in some arbitrary position within the structure itself. In most cases, a node lies on an edge or corner of an element and is used to ensure the compatibility of displacements between elements or between the model and a support boundary.