ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE
ENGINEERING AND TECHNOLOGY

DEVELOPING A 3D FINITE ELEMENT SOFTWARE WITH AN OBJECT ORIENTED APPROACH

M.Sc. THESIS

Halid Eren ADAK

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

JANUARY 2014

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

DEVELOPING A 3D FINITE ELEMENT SOFTWARE WITH AN OBJECT ORIENTED APPROACH

M.Sc. THESIS

Halid Eren ADAK
(511101134)

Department of Aeronautical and Astronautical Engineering

Aeronautical and Astronautical Engineering Programme

Thesis Advisor: Prof. Dr. Zahit MECİTOĞLU

JANUARY 2014

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

NESNE YÖNELİMLİ PROGRAMLAMA YAKLAŞIMI İLE ÜÇ BOYUTLU SONLU ELEMANLAR YAZILIMI GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Halid Eren ADAK
(511101134)

Uçak ve Uzay Mühendisliği Anabilim Dalı

Uçak ve Uzay Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Zahit MECİTOĞLU

OCAK 2014

Halid Eren ADAK, a **M.Sc.** student of ITU **Graduate School of Science Engineering and Technology** student ID **511101134**, successfully defended the **thesis** entitled "**DEVELOPING A 3D FINITE ELEMENT SOFTWARE WITH AN OBJECT ORIENTED APPROACH**", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Prof. Dr. Zahit MECİTOĞLU** ....
İstanbul Technical University

Jury Members : **Prof. Dr. Mehmet Hakkı OMURTAG** ....
İstanbul Technical University

Assoc. Prof. Dr. Vedat Ziya DOĞAN ....
İstanbul Technical University

Date of Submission : 17 December 2013
Date of Defense : 23 January 2014

To my family,

FOREWORD

I would like to express my greatest appreciation and gratitude to my supervisor Prof. Dr. Zahit Mecitoğlu for his support and guidance throughout the preparation of this thesis.

I owe developers of Open Cascade library my special thanks since I wouldn't be able to complete this thesis without their valuable open source library. Additional thanks go to the authors of Gmsh for their unique mesh generator library.

Finally, my sincere appreciation is extended to my mother, Gülseren, my father, Dr. Burhan, and all my brothers and sister for their tireless patience, continuous encouragement and great understanding during the whole of my graduate training and this thesis.

December 2013 Halid Eren ADAK
(Industrial Engineer)

TABLE OF CONTENTS

Page

FOREWORD .... **ix**
TABLE OF CONTENTS .... **xi**
ABBREVIATIONS .... **xiii**
LIST OF TABLES .... **xv**
LIST OF FIGURES .... **xvii**
1. INTRODUCTION **1**
1.1. Objective and Scope 1
1.2. Literature Review 2
1.3. Organization 3
2. DEVELOPMENT PROCEDURE **5**
2.1. Object-Oriented Programming Philosophy 5
2.2. Unified Modelling Language 6
2.3. EAFE Software Development Procedure 8
2.4. Programming Language Selection 9
2.4.1. Programming in Java 10
2.4.2. Programming in C++ 10
2.5. Integrated Development Environment (IDE) 10
2.6. Windows Programming with the Microsoft Foundation Classes (MFC) 11
3. GEOMETRY MODULE **13**
3.1. Computer Graphics 13
3.2. 3D Computer Graphics and OpenGL 14
3.3. Geometry Kernel 18
3.4. Open Cascade Technology (OCCT) 19
3.4.1. Modeling module 19
3.4.2. Visualization module 19
3.4.3. Data exchange module 19
3.4.4. Application framework module 20
3.5. Implementation of OCCT Modules in EAFE Software 20
3.6. Open Cascade Application Framework (OCAF) 21
4. MESH MODULE **25**
4.1. Mesh Generators 25
4.2. GMSH Mesh Framework 26
4.3. Implementation of Gmsh in EAFE 26
5. SOLVER MODULE **31**
5.1. Object-Oriented Programming 31
5.1.1. Fundamental concepts in object-oriented programming 31
5.1.1.1. Object 31
5.1.1.2. Class 31
5.1.1.3. Encapsulation 32

5.1.1.4. Method 33
5.1.1.5. Inheritance 33
5.1.1.6. Polymorphism 34
5.2. Finite Element Method 34
5.2.1. Three-dimensional stress analysis 35
5.2.1.1. Fundamental equations 35
5.2.1.2. Tetrahedral element (Tet-4) 37
5.2.1.3. Stress calculations 42
5.2.1.4. Dynamic consideration 43
5.3. Object-Oriented Finite Element Analysis 46
5.3.1. EafeLib: A C++ finite element analysis library and its base classes 46
5.3.1.1. Element class 47
5.3.1.2. Node class 47
5.3.1.3. Material class 49
5.3.1.4. Load class 51
5.3.1.5. Boundary condition class 52
5.3.2. Global stiffness and mass matrices assembly process 52
5.3.2.1. Model class 52
5.3.3. Linear algebra library 55
5.3.3.1. PETSc 55
5.3.3.2. Trilinos 56
5.3.4. Input and output file formats 56
5.3.4.1. Input file 56
5.3.4.2. Output file 58
6. RESULTS AND DISCUSSION 61
6.1. Application Tests 61
6.1.1. A loaded cantilever beam 61
6.1.2. A plate with a hole 64
6.1.3. A support beam with a uniform pressure 67
6.1.4. Dynamic analysis of a cantilever beam 70
7. CONCLUSIONS 73
REFERENCES 75
APPENDICES 79
APPENDIX A 81
APPENDIX B 83
APPENDIX C 90
APPENDIX D 96
CURRICULUM VITAE 99

ABBREVIATIONS

API : Application Programming Interface
CAD : Computer Aided Design
CAE : Computer Aided Enginering
CAM : Computer Aided Manufacturing
DEAL : Differential Equations Analysis Library
FEM : Finite Element Method
GPU : Graphics Processing Unit
GUI : Graphical User Interface
IDE : Integrated Development Environment
MFC : Microsoft Foundation Classes
MPI : Message Passing Interface
OCAF : Open Cascade Application Framework
OCCT : Open CasCade Technology
OOFEM : Object-Oriented Finite Element Modeling
OOP : Object-Oriented Programming
OpenGL : Open Graphics Library
OS : Operating System
PC : Personal Computer
RAD : Rapid Application Development
UML : Unified Modeling Language

LIST OF TABLES

Page

Table 2.1 : UML diagrams. 7
Table 2.2 : Softwares with implementation language and target operating system [19]. 9
Table 3.1 : CAD/CAE softwares and related geometry kernels. 18
Table 5.1 : Open source linear algebra packages. 56
Table 6.1 : Displacements and stress results with different element numbers.......... 64
Table 6.2 : Displacements and stress results with different element numbers.......... 67
Table 6.3 : Displacements and stress results with different element numbers.......... 70
Table 6.4 : Cantilever beam mode shapes. 71
Table 6.5 : Cantilever beam natural frequencies. 72

LIST OF FIGURES

Page

Figure 2.1 : A UML diagram example [16]. 7
Figure 2.2 : EAFE software structure with dependent libraries. 9
Figure 2.3 : MFC Document/View concept [22]. 11
Figure 2.4 : Document/View structure. 12
Figure 3.1 : 3D perception: (a)How you see three dimension. (b)A simple wireframe 3D cube. 15
Figure 3.2 : Utah Teapot: (a)Wireframe model. (b) A modern render model. (c) Original teapot [28]-[30]. 15
Figure 3.3 : An example of a simple OpenGL function. 16
Figure 3.4 : Screenshot of rendered polygon. 17
Figure 3.5 : Use of normal vectors in light calculation [32]: (a) Normal vectors perpendicular to rectangles. (b) Normal vectors perpendicular to surface. 17
Figure 3.6 : Geometry module of EAFE software. 21
Figure 3.7 : Import export properties of EAFE software. 21
Figure 3.8 : A basic OCAF data framework. 23
Figure 3.9 : EAFE software's OCAF based data framework. 24
Figure 4.1 : EAFE software with GMSH mesh framework. 27
Figure 4.2 : Gmsh library usage. 28
Figure 4.3 : A solid shape with different mesh options: (a) Solid shape. (b) Constant mesh size. (c) Variable mesh size. 29
Figure 5.1 : A simple code fragment to show classes and objects. 32
Figure 5.2 : An example class structure. 32
Figure 5.3 : A simple code fragment to show method usage. 33
Figure 5.4 : An inheritance hierarchy example. 34
Figure 5.5 : Three dimensional stresses on an element [41]. 36
Figure 5.6 : Master element used in shape functions. 38
Figure 5.7 : The simplified UML class diagrams of EafeLib solver. 48
Figure 5.8 : The UML representation of the element class. 49
Figure 5.9 : The UML representation of the node class. 50
Figure 5.10 : The UML representation of Material and IsotropicMaterial classes... 51
Figure 5.11 : The UML representation of the Load and the DistributedLoad classes. 51
Figure 5.12 : The UML representation of the Boundary Condition classes. 52
Figure 5.13 : The UML representation of the SparseModel class. 53
Figure 5.14 : The UML activity diagram of the assembler function. 54
Figure 5.15 : EafeLib solver input file format. 57
Figure 5.16 : EafeLib solver displacements output file format. 58
Figure 5.17 : EafeLib solver stress output file format. 59
Figure 6.1 : A cantilever beam with a uniform load. 61

Figure 6.2 : Cantilever beam deformation contours in EAFE. 62
Figure 6.3 : Cantilever beam deformation contours in Abaqus. 62
Figure 6.4 : Cantilever beam Mises stress contours in EAFE.................................63
Figure 6.5 : Cantilever beam Mises stress contours in Abaqus. 63
Figure 6.6 : A plate with a hole. 64
Figure 6.7 : Plate deformation contours in EAFE..65
Figure 6.8 : Plate deformation contours in Abaqus..65
Figure 6.9 : Plate Mises contours in EAFE...66
Figure 6.10 : Plate Mises contours in Abaqus...66
Figure 6.11 : Support beam..67
Figure 6.12 : Support beam displacement contours in EAFE..................................68
Figure 6.13 : Support beam displacement contours in Abaqus................................68
Figure 6.14 : Support beam Mises contours in EAFE. 69
Figure 6.15 : Support beam Mises contours in Abaqus. 69
Figure 6.16 : A cantilever beam 70
Figure A.1 : Solid model of cantilever beam and its finite element mesh. 81
Figure A.2 : Defining boundary condition and load for the model..........................82
Figure B.1 : Adding points and lines to create half of the plate...............................83
Figure B.2 : Creating a wire by connecting consecutive lines and an arc. 84
Figure B.3 : Using mirror function with an axis to complete the frame of the plate.85
Figure B.4 : Creating two faces and extruding them along an axis. 86
Figure B.5 : Fusing two separate halves and defining material for the plate............87
Figure B.6 : Discretizing the plate by increasing mesh density on critical region....88
Figure B.7 : Defining boundary condition and load for the model..........................89
Figure C.1 : Adding points and lines to create the cross-section of the model.........90
Figure C.2 : Creating a face and an axis for extrusion...91
Figure C.3 : Extruding the face along the axis and creating cylinders for the holes.92
Figure C.4 : Cutting cylinders from the part and defining material for the model...93
Figure C.5 : Discretizing the model and adding boundary conditions. 94
Figure C.6 : Adding distributed load. 95
Figure D.1 : Solid model of cantilever beam and its finite element mesh. 96
Figure D.2 : Defining boundary condition for the model. 97

DEVELOPING A 3D FINITE ELEMENT SOFTWARE WITH AN OBJECT ORIENTED APPROACH

SUMMARY

In this thesis, a 3D finite element software is developed in the basis of an object-oriented approach. Most of the problems in engineering fields are modeled by using computers, and these models are solved by using various numerical methods. One of the most frequently used numerical methods is the finite element method. The finite element method is a powerful numerical technique for finding approximate solutions of partial differential equations as well as of integral equations. The basic concept in the physical interpretation of the finite element method is the subdivision of the mathematical model into disjoint (non-overlapping) components of simple geometry called finite elements or elements for short. The response of each element is expressed in terms of a finite number of degrees of freedom characterized as the value of an unknown function, or functions, at a set of nodal points.

Programs that implements finite element method in computers have long been written in procedural languages such as FORTRAN and C. However, for the last twenty years developers who seek to improve finite element programs modularity, extensibility, and maintainability have a growing interest in developing finite element software with object-oriented programming approach.

The software developed in this thesis, EAFE, is written in C++ language with an object-oriented approach. The preferred integrated development environment is Microsoft Visual Studio. The target operating system is Microsoft Windows and, therefore, Microsoft Foundation Classes (MFC) is used to develop the graphical user interface.

EAFE software has three main modules. The first module, which is developed by using open source Open Cascade library, is the geometry module and it is used to build 1D, 2D, or 3D geometric models. The second module, which is developed by using open source Gmsh library, is the mesh module and it is used to discretize a given geometric domain. The third module is the solver module and it is used to assemble global stiffness matrix, global mass matrix, and global force vector and to solve the system of linear equations. Different from the other two modules a stand-alone library named EafeLib is developed from scratch for the solver module.

EafeLib library contains a number of C++ classes designed to do finite element analysis in 3D with an object-oriented approach. It is built around six main classes: Node, Element, Load, BoundaryCondition, Material, and Model. It also has some auxiliary classes such as InputReader, Solver, and OutputWriter. The primary class in EafeLib solver is the Model class.

Some benchmark problems are solved by making use of developed EafeLib library and it is shown that object-oriented programming approach is well suited for implementing finite element method in computer.

NESNE YÖNELİMLİ PROGRAMLAMA YAKLAŞIMI İLE ÜÇ BOYUTLU SONLU ELEMANLAR ANALİZİ YAZILIMI GELİŞTİRİLMESİ

ÖZET

Bu çalışmada nesne yönelimli programlama yaklaşımı ile, üç boyutlu sonlu elemanlar analizi gerçekleştirebilecek bir yazılım geliştirilmiştir. Günümüzde karşılaşılan mühendislik problemlerinin neredeyse tamamı bilgisayar ortamında modellenmekte ve çözümlerinde sayısal yöntemlerden istifade edilmektedir. Bu nümerik yöntemlerden en sık kullanılanı sonlu elemanlar yöntemdir. Geleneksel olarak sonlu elemanlar yöntemi için geliştirilen algoritmaların çoğunda FORTRAN ve C gibi prosedürel programlama dilleri kullanılmaktaydı. Prosedürel programlama dillerinin sonlu elemanlar analizi için sağladığı en önemli avantaj performanstır. Ancak bu yazılımların bilgisayar endüstrisindeki gelişmelere paralel olarak büyümesi ve karmaşıklaşması bakım ve modifikasyon maliyetlerinin artmasına sebep olmuş ve yazılımcıları farklı programlama yaklaşımları kullanmaya zorlamıştır. Son 20 yıldır akademide ve endüstride sonlu elemanlar analizi yazılımlarına esneklik kazandırabilmek için nesne yönelimli programlama yaklaşımı ile geliştirilmesi düşüncesine artan bir ilgi söz konusudur.

Bilindiği üzere günümüzde en sık kullanılan nesne yönelimli programlama dilleri C++ ve Java'dır. Yazılımın geliştirilme sürecinde ihtiyaç duyulabilecek kütüphanelerin neredeyse tamamının C++ dilinde geliştirildiği gerçeği göz önünde tutularak yazılımın geliştirilmesinde C++ programlama dili kullanılmasına karar verilmiştir. Derleyici olarak ise Microsoft Visual Studio tercih edilmiştir. Hedef işletim sistemi Microsoft Windows olarak belirlendiğinden kullanıcı arayüzü için Microsoft Foundation Classes (MFC) kütüphanelerinden faydalanılmıştır.

Geliştirilen yazılım üç ana bölümden oluşmaktadır. Bu bölümlerden ilki bir, iki, veya üç boyutlu model oluşturmak için kullanılabilecek bir geometri modülü, ikincisi oluşturulan herhangi bir geometrik modeli basit geometrili elemanlara bölebilecek bir çözüm ağı modülü, üçüncüsü ise sonlu elemanlar çözümünü gerçekleştirebilecek bir çözücü modülüdür. Bunlara ek olarak problemin çözümü ile elde edilen sonuçlar kullanıcı tarafından daha rahat yorumlanabilmesi için bir renk dağılımı şeklinde görselleştirilmektedir. Geometri modülünün geliştirilmesinde açık kaynak kodlu Open Cascade kütüphanesinden, mesh modülünün geliştirilmesinde ise aynı şekilde açık kaynak kodlu Gmsh kütüphanesinden yararlanılmıştır. Buna karşılık çözücü modülünde kullanılan kütüphane nesne yönelimli programlama yaklaşımı ile sıfırdan yazar tarafından geliştirilmiştir ve söz konusu programa entegre edilmiştir.

Günümüzde yaygın olarak kullanılan bilgisayar destekli tasarım programlarının tamamı geometrik çekirdek adı verilen ve genel anlamda bilgisayarda geometrik şekiller çizmeye yardımcı olacak fonksiyonları içeren kütüphaneler kullanılarak geliştirilir. Bu geometrik çekirdeklerin en bilinenleri Spatial firmasına ait ACIS ve Siemens firmasına ait Parasolid kütüphaneleridir. Lisans bedelleri ödemek suretiyle

kullanılabilecek bu kütüphanelere alternatif olarak bu çalışmada açık kaynak kodlu sunulan Open Cascade kütüphanesinden istifade edilmiştir.

Geometri modülü geometrik modellemede temel olarak kullanılabilecek nokta, çizgi, yay gibi bir takım basit geometrik şekillerin çizilebileceği fonksiyonlar ile birlikte dikdörtgenler prizması, silindir ve küre gibi temel katı cisimlerin kolayca eklenebileceği kısa yolları içeren bir ortam olarak geliştirilmiştir. Bunlara ek olarak bir yüzeye kalınlık vererek katı oluşturma, bir yüzeyi bir eksen etrafında döndürerek katı oluşturma, iki katı modeli birleştirip yeni bir katı oluşturma ve bir katıdan bir başka katı çıkararak katı oluşturma gibi bilgisayar destekli tasarımın en temel fonksiyonları da bu modülde yer almaktadır. Geliştirilen yazılımın geometri modülüne "import" ve "export" fonksiyonları eklenerek farklı programlar ile oluşturulan ve yaygın olarak kullanılan IGES ve STEP gibi formatlarda kaydedilen geometrik modellerinde yazılımda açılıp analiz edilebilmesine olanak sağlanmıştır.

Çözüm ağı oluşturma modülü sonlu elemanlar analizinin en önemli aşamalarından biri olan geometrik şekillerin küçük sonlu elemanlara bölünerek çözüm ağı geliştirilmesi işleminin yapılabilmesi için programa eklenmiştir. Bu işlem sırasında kullanılan eleman çeşitleri tek boyutlu çizgisel eleman, iki boyutlu düzlem eleman olarak üçgen eleman ve üç boyutlu katı eleman olarak da dörtyüzlü elemandır. Bu modülün önemli özelliklerinden biri istenildiği takdirde söz konusu geometrinin kritik bölgelerindeki ağ sıklığının arttırılabilmesine imkan tanımasıdır.

Çözücü modülü genel anlamda, model, düğüm noktası, eleman, sınır koşulları, yük ve malzeme adlı altı temel sınıf ve bunlara ek olarak girdi okuyucu, çıktı yazıcı ve çözücü gibi yardımcı sınıflar kullanılarak, nesne yönelimli programlama yaklaşımıyla C++ dilinde geliştirilen bir sonlu elemanlar analizi kütüphanesidir. Bu kütüphane geometrisi tanımlanmış, malzemesi belirlenmiş, çözüm ağı geliştirilmiş, sınır koşulları ve yükleri girilmiş bir modeli önceden belirlenmiş bir girdi dosyası formatında alıp sonlu elemanlar yöntemi kullanarak çözmek ve elde edilen sonuçları aynı şekilde önceden formatı belirlenmiş bir çıktı dosyası halinde sunmak için geliştirilmiştir. Girdi dosyası geliştirilen kütüphanenin girdi okuyucusu sınıfı yardımıyla okunur. Dosyadaki bilgiler ışığında kütüphanenin düğüm noktası sınıfı kullanılarak modelde bulunan her bir düğüm noktası için düğüm noktası numarası ile x, y, ve z kartezyen kordinatları bilgisini barındıran düğüm noktası nesneleri oluşturulur. Modelde bulunan her bir malzeme girdisi için ise kütüphanenin malzeme sınıfı kullanılarak malzeme numarası, Elastisite modülü, Poisson oranı ve yoğunluk değişkenlerini içeren malzeme nesneleri oluşturulur. Modelde bulunan her bir yük girdisi için de kütüphanenin yük sınıfı kullanılarak yük numarası, yük şiddeti ve yük doğrultusu değişkenlerini içeren yük nesneleri oluşturulur. Benzer şekilde modelde tanımlanmış her bir sınır koşulu girdisi için kütüphanenin sınır koşulları sınıfı kullanılarak x, y, ve z yönündeki u, v, ve w değişkenlerini içeren sınır koşulu nesneleri oluşturulur. Bu işlemler gerçekleştikten sonra oluşturulan bu nesneler kütüphanenin model sınıfı aracılığı ile girdi dosyasındaki her bir eleman girdisine karşılık gelecek eleman nesnelerini oluşturmak için kullanılır. Dosyadaki bir elemenan girdisi için düğüm noktası nesneleri, malzeme nesneleri, yük nesneleri ve sınır koşulları nesneleri kullanılarak bir eleman nesnesi oluşturulur. Bu eleman nesnesinin katılık matrisi hesaplanır ve bu matris global katılık matrisi içine yerleştirilir. İhtiyaç duyuluyorsa elemanın kütle matrisi hesaplanıp global kütle matrisi içine yerleştirilir. Eleman nesnesi bellekte gereksiz yer tutmaması için bu işlemden sonra silinir ve bu adımlar modeldeki her bir eleman için tekrar eder. Böylelikle global katılık matrisi ve global kütle matrisi hazırlanmış olur.

Burada bahsi geçen matrislerin hazırlanması için gereken matris sınıfları C++ dilinde standart olarak bulunmadığından bu matris sınıfları ya kullanıcı tarafından sıfırdan geliştirilmeli yada daha önce geliştirilmiş hazır matris sınıflarından istifade edilmelidir. Bu çalışmada hem sıfırdan EafeMatrix adı verilen temel bir C++ matris sınıfı geliştirilmiş hemde uzman bir ekip tarafından geliştirilmiş açık kaynak kodlu Trilinos adlı C++ lineer cebir kütüphanesinden faydalanılmıştır. Geliştirilen yazılımın bütünlüğünün bozulmaması ve bu yazılım üzerinde çalışacak herhangi bir geliştiricinin, yazılıma yeni eleman tipleri eklemek istemesi durumunda karşılaşacağı, harici kütüphanelerin kullanılmasından kaynaklanacak yabancılığın önüne geçilmesi için eleman katılık matrisleri yazılımın içerisindeki EafeMatrix sınıfından türetilmiştir. Global katılık matrisinde ise oluşturulacak matrisin, lineer denklem takımlarının çözümü için kullanılacak Trilinos çözücü sınıflarının girdi olarak alabileceği bir formatta olması gerekliliği göz önünde tutularak Trilinos kütüphanesinin seyrek matris sınıfı kullanılmıştır. Daha sonra bu matrisler global yük vektörü ile birlikte çözücü sınıfında lineer denklem takımlarının çözülüp sonuçların elde edilmesi için kullanılır. Son olarak elde edilen düğüm noktası yer değiştirmesi veya eleman von Mises gerilme değerleri gibi sonuçların kullanıcılar tarafından rahat yorumlanabilmesi için bir renk dağılımı şeklinde görselleştirilmesi işlemi gerçekleştirilir.

Geliştirilen EAFE yazılımı kullanılarak elde edilen sonuçların doğruluğunu test etmek için statik ve dinamik bazı örnek problemler çözülmüş ve sonuçlar Abaqus yazılımı ile karşılaştırılmıştır. Her iki yazılım ile elde edilen sonuçların büyük oranda örtüştüğü gösterilmiştir. Böylelikle nesne yönelimli programlamanın sonlu elemanlar analizi için uygun bir yaklaşım olduğu bu kütüphane vasıtasıyla gösterilmiştir.

1. INTRODUCTION

Many physical phenomena in science and engineering are mathematically modelled by using partial differential equations. These equations have long been solved by using analytical methods. However, due to the difficulty and inadequacy of analytical methods in complex field problems, there has been an interest to develop different numerical methods instead. One of the most powerful and widely used numerical methods for finding solutions to such problems is the Finite Element Method (FEM).

The computations in the FEM are generally long and tedious, therefore requires a computer. The use of computers in the FEM programming is shown a parallel growth with the developments in the computer industry such as increasing processor capabilities and the introduction of personal computers (PC). Nowadays FEM programs has become commonplace and even a simple PC can be used to obtain solutions to very complicated problems.

As it is the case in most scientific application, traditionally there has been a tendency to write FEM codes, because of their performance, in procedural languages such as FORTRAN. However, in the last decade there has been a shift from procedural languages to object-oriented languages such as C++ and Java. Object-Oriented Programming (OOP) is a programming methodology based on objects, instead of just functions or procedures and it is shown that OOP is well suited for FEM programming.

1.1. Objective and Scope

The focus of this thesis is to develop a three-dimensional, object-oriented finite element analysis software. The object-oriented environment selected with the intention of providing efficient, robust, modular, and extensible finite element code structure for future development. An important aspect of the work is the development of a modern graphical user interface (GUI) which incorporates a geometry module

for creating and manipulating 3D solid shapes, a mesh module for discretizing a given domain, and a solver module to solve the linear system of equations.

The geometry module is built with Open Cascade, an open source geometry kernel, and the mesh module is built with Gmsh, an open source mesh framework. The solver module is developed in C++ language with an object-oriented approach and these three modules incorporated in a GUI, which is developed in Microsoft Visual Studio 2010 Ultimate by using Office Ribbon Interface tools.

1.2. Literature Review

Over the last 20 years, some work has been done towards developing finite element analysis programs with an object oriented programming approach. The pioneers of the object-oriented finite element programming idea are Fenves [1] who highlighted the potential benefits of using object-oriented programming approach in engineering software, Rehak [2] who considered the subject from a knowledge-engineering perspective, Peskin and Russo [3] who organized three base classes: Problem, Domain and Equation to solve partial differential equations, and Miller [4] who utilized Degree-of-freedom, Node, and Element classes in his work.

The first detailed description of applying object-oriented programming to the finite element method is provided by Forde and co-workers [5] to deal with linear two-dimensional problems in solid mechanics. They put forward the base classes of object-oriented finite element analysis such as Element, DispBC, ForceBC, Material, and Dof. These classes have been reused by several authors to organize their program structures. Likewise, in early papers, Zimmerman, et al. [6]-[9] have favored object-oriented programming over procedural programming and studied its applications to the finite element method by using Smalltalk and C++ languages. They considered linear dynamic analysis in their work by using three groups of classes. The first group contains the finite element classes such as Node, Element, Load, Material, etc., the second group is a collection of assistant classes like GaussPoint, Polynomial, etc., and the third group is a gathering of data storage classes such as Array, Matrix, etc. In addition, by redefining some of the original classes such as Domain, Element, and Material from their previous work, they considered nonlinear finite element analysis as well.

Lu, et al. [10],[11] contributed to the field by developing an object-oriented finite element code called FE++. Their approach differs from the others in the way they handled the assembly process by making use of a central Assemble object. Another contribution they made is a complete C++ linear algebra library as an alternative to the standard FORTRAN library LAPACK.

Bangerth et al. [12], [13] developed a flexible and efficient object-oriented library called Differential Equations Analysis Library (DEAL) II in which they work towards the computational solution of partial differential equations using adaptive finite elements.

Patzák et al. [14] developed yet another program called Object Oriented Finite Element Modeling (OOFEM). Their aim was to develop not only an efficient and robust tool for FEM computations but also a modular and extensible environment for future development.

In addition to the works mentioned here, there are several other commercial or public domain object-oriented finite element analysis libraries developed by researchers and engineers from academy and industry.

1.3. Organization

This thesis contains seven chapters including an introduction.

In Chapter Two an overview of the object-oriented programming and the unified modeling language is provided, the development procedure of EAFE software along with the preferred programming language and the integrated development environment is presented.

In Chapter Three an introduction to the computer graphics and the OpenGL application programming interface is given to illustrate the necessity to employ a geometry kernel in the development of EAFE software's geometry module. In addition, the selected open source geometry kernel: Open Cascade, and its application framework is detailed.

In Chapter Four an outline of mesh generation and freely available mesh generators, or mesh frameworks, is given. In addition, the Gmsh library, which is the mesh generator used in EAFE software's mesh module, is presented.

In Chapter Five the development of a processor, named EafeLib, for finite element analysis with object oriented programming approach is provided. Moreover, the UML representation of the base classes in the EafeLib processor is given in detail.

In Chapter Six some example problems are solved to check the accuracy of the results obtained from the developed software by comparing them with the results from commercial FEA software.

In Chapter Seven conclusions are made and further work is discussed.

2. DEVELOPMENT PROCEDURE

2.1. Object-Oriented Programming Philosophy

Object-Oriented Programming (OOP) concepts were first introduced by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Centre in Oslo in the early 1960s. They developed a programming language for discrete event simulation called SIMULA in which OOP concepts such as objects, classes, inheritance, etc., were used [15]. Many object-oriented programming languages developed later including Smalltalk, LISP, Object Pascal and C++ are based on the ideas of SIMULA language. Object-Oriented programming has become an indispensable programming methodology for large software systems.

C and FORTRAN are examples of procedural languages in which functions, or routines, are the means of programming. In procedural programming small functions, or subroutines, are written to complete simple tasks. These small functions are brought together in large functions to accomplish complex tasks. A program developed in procedural programming concept is nothing but a collection of these large functions arranged in a way that upon instruction computer can perform them in sequence. Even though this approach can be used to develop small computer programs, for large software with thousands of routines and subroutines, software modification, maintenance or extensions becomes virtually impossible.

Object-Oriented Programming is the result of the growing demand for a new approach to develop flexible, modular and reusable software components to meet the requirements of dynamic and competitive environment. In order to develop the most natural way of programming OOP offers the concept of a self-sustainable object which is inspired from the real world objects such as people, animals, cars, buildings, computers and so on. In real world, objects have attributes like size, mass, height, color and they all exhibit behaviours like a truck accelerates, brakes, turns, or a person walks, runs, sleeps and works. Moreover, objects in real world communicate with each other and can be gathered according to their attributes and behaviours. For

example, cars, buses, trucks all shares some attributes, exhibit similar behaviours, and might be grouped under vehicles.

Object-Oriented Programming uses objects initiated from prototype packages of data types called classes, similar in many ways to the real world objects. They have attributes, called data members, and behaviours, called methods, that can either view or manipulate these hidden data members. In addition, OOP objects also communicate with each other in terms of messages they send or receive and taking advantage of the objects common attributes and behaviours new classes are easily created through a process called inheritance. Fundamental concepts in Object-Oriented Programming such as inheritance, encapsulation, and polymorphism are explained in detail, along with their implementation to the Finite Element Method in Chapter 5.

The notion of using objects as building blocks for software development in Object-Oriented Programming has become successful to a large extent in contributing software's modularity, reusability, and maintainability.

2.2. Unified Modelling Language

Unified Modeling Language (UML) is a graphical modelling language developed by Grady Booch, Ivar Jacobson and James Rumbaugh to support Object-Oriented software analysis and design. UML was adopted by Object Management Group in 1997 and has become an international industry standard for modeling software intensive systems.

A simple UML diagram with shape, color, rectangle and circle classes is shown in Figure 2.1. Each of the rectangular boxes in the figure is the UML's graphical representation of a class. In this rectangular box, the class name is written on the top, data members (attributes) are written in the middle, and member functions (behaviours) are written at the end. The triangle between rectangle class and shape class shows an "is-a" relationship. This relationship refers to the fact that each rectangle object is also a shape object. This means rectangle class, which is a derived class, inherits data members and member functions of shape class, which is a base class in this case. Constructing classes in this manner is called inheritance and it is one of the most important features of Object-Oriented Programming. The use of

inheritance during a new class generation provides code re-use, due to the fact that derived classes by default retains member functions of base classes.

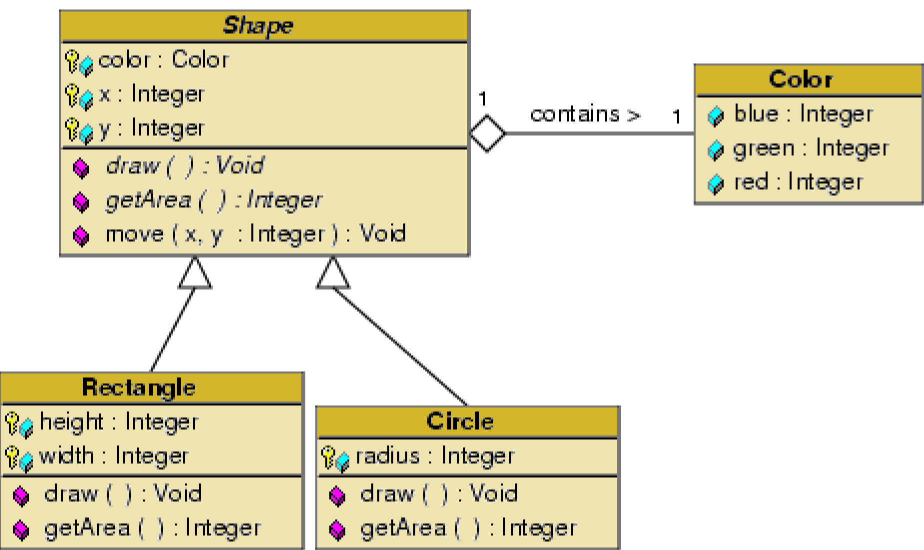


Figure 2.1 : A UML diagram example [16].

On the other hand, the white diamond shape between color class and shape class shows a "has-a" relationship. This relationship illustrates the fact that a color object is used as a data member in each shape object. Constructing new classes by using existing classes as data members is called aggregation.

The UML version 2.2 contains 14 different diagrams under two main categories: structural diagrams and behavioral diagrams, as detailed in Table 2.1. Structural diagrams are used to show the static parts of the modeled system, while behavioral diagrams are used to show the dynamic activities of the system [17]. In this thesis, UML's structural and behavioral diagrams are used to represent the developed EAFE Software.

Table 2.1 : UML diagrams.

UML Diagrams	
Structural UML Diagrams	Behavioral UML Diagrams
Class diagram	Activity diagram
Component diagram	Communication diagram
Composite structure diagram	Interaction overiew diagram
Deployment diagram	Sequence diagram
Object diagram	State diagram
Package diagram	Timing diagram
Profile diagram	Use case diagram

2.3. EAFE Software Development Procedure

The requirements analysis is the first and most important step in the development process of every software system. For basic 3D finite element analysis software, which is the subject of this thesis, the requirements are gathered as follows:

- It should have a geometry module (CAD) in which a user is able to create some simple solid parts. In addition, there should be an import and export functionality to exchange geometric shapes in standard formats.
- It should have a mesh module in which a user can generate at least an unstructured mesh for the geometry with triangular or tetrahedral elements, and change mesh density on critical regions.
- It should have a section that supports adding loads to and defining boundary conditions for the system.
- It should have a solver module in which global stiffness matrix and global force vector is assembled, global linear equation system is solved and nodal displacement values are found.
- It should have a post-processing module in which the results such as nodal displacement and element stress values can be displayed.
- It should have an easy to use graphical user interface (GUI) so that a user can interact with the mentioned modules without any difficulty.

Designing software to meet all of these requirements is a formidable and time-consuming task. To facilitate and accelerate the design process it has been taken advantage of some ready-to-use open source libraries. In the development of EAFE software's geometry module Open Cascade library is used and explained in detail in Chapter 3. While in mesh module Gmsh library is used and described in Chapter 4. For graphical user interface Microsoft Foundation Classes (MFC) are used.

On the other hand, for EAFE software's solver module, an FEM solver library named EafeLib has been written from scratch and a complete explanation is given in Chapter 5. The structure of EAFE software with its three main modules and related libraries is illustrated in Figure 2.2.

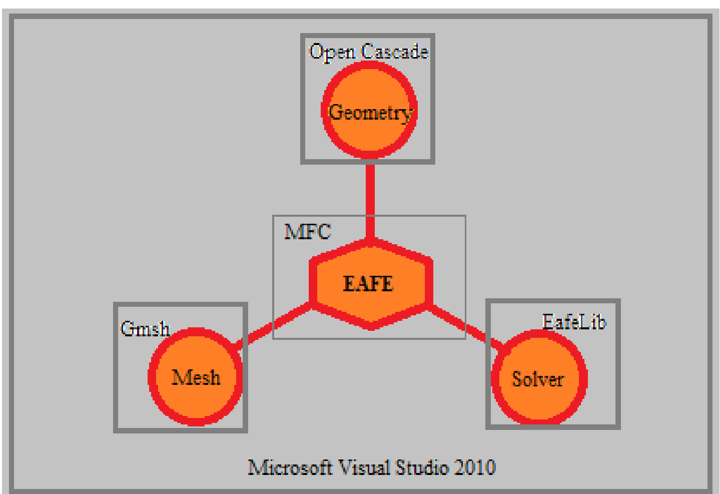


Figure 2.2 : EAFE software structure with dependent libraries.

2.4. Programming Language Selection

Traditionally there has been a tendency to develop finite element codes in procedural languages such as FORTRAN or C, due to the computational speed and ease of implementation these languages have. Even though there are still numerous codes written in FORTRAN, there has been a shift for the last 20 years from procedural languages to Object-Oriented Languages such as C++ and Java [18]. Table 2.2 shows some well-known open source finite element analysis software, languages used in their development and the target operating systems.

Table 2.2 : Softwares with implementation language and target operating system [19].

Software	Language	Operating System
CalculiX	Fortran	GNU/Linux, Windows
Code Aster	Python and Fortran	GNU/Linux, FreeBSD
Deal.II	C++	GNU/Linux, Unix, Mac OS X, Windows
DUNE	C++	GNU/Linux, Unix, Mac OS X
FEniCS Project	Python and C++	Linux, Unix, Mac OS X, Windows
FreeFem++	C++	GNU/Linux, Mac OS X, Windows, Solaris
Impact	Java	Linux, Windows

C++ and Java are the most widely used languages for the Object-Oriented Programming. The advantages, disadvantages, strong and weak sides of these two languages are explained in the following two sections.

2.4.1. Programming in Java

Java is a strictly object-oriented computer programming language developed by James Gosling at Sun Microsystems and announced in 1995. Java uses a syntax similar to C++ syntax. In order to prevent memory leaks, manual memory allocation and deallocation is eliminated in Java with the addition of a Garbage collector. Java source codes are compiled to java bytecodes. Perhaps one of the most important features of the Java language is the Java Virtual Machine that can execute Java bytecodes. Making use of bytecodes and virtual machines Java provides a software portability mechanism called “write once, run anywhere”, which means when a Java program compiled, it can be run on almost any device that has a Java Virtual Machine. However, there is a performance loss due to the introduction of virtual machine as an intermediate step to run the compiled program. Even though Just-in-time compilation was introduced to boost the performance of Java programs, compared to C++, it is less frequently used for programming finite element method [20].

2.4.2. Programming in C++

C++ is a popular software development language that is widely used for both commercial and academic purposes. It is developed by Bjarne Stroustrup in 1980 at Bell Laboratories. C++ is based on C language, and in addition to C capabilities, it has also support for object-oriented programming. Different from Java language C++ supports multiple inheritance and operator overloading [21].

In the development of the EAFE 3D finite element software, C++ language is used because it is robust, fast, and reliable. Besides, most of the libraries required to develop finite element software such as geometry kernels, mesh frameworks, and linear algebra packages are generally written in C++ language.

2.5. Integrated Development Environment (IDE)

Integrated development environments or interactive development environments are programs specifically designed to assist software developers in the software development, maintenance and modification processes. IDEs are generally composed of four components: a graphical user interface (GUI) builder, a source code editor, a

compiler, and a debugger. GUI builder and source code editor are used to develop programs, compiler is used to translate written source codes to object codes, and debugger is used to locate and fix probable bugs in the software. Some of the famous IDEs are Microsoft Visual Studio, Oracle Netbeans, Xcode, and Eclipse. EAFE software's target operating system is Microsoft Windows OS. Hence, it has been developed in Microsoft Visual Studio 2010 Professional IDE.

2.6. Windows Programming with the Microsoft Foundation Classes (MFC)

Microsoft Foundation Class Library is a collection of C++ wrapper classes for Windows Application Programming Interface (API). It is a framework to develop Windows based applications. In MFC, there is a certain structure for processing and storing application data that must be used by the developer to develop an MFC based application. Although it looks restrictive, the advantages of using this structure far outweigh any possible disadvantage. This structure is based on document and view objects. A document object is an instance of application specific document class that is created by extending MFC's CDocument class. Every data member that is used in the application must be stored in this document object. On the other hand, a view object is an instance of an application specific view class that is created by extending MFC's CView class. View objects are used to display the data stored in the document object. In Figure 2.3 the data that a document object contains displayed with two view objects.

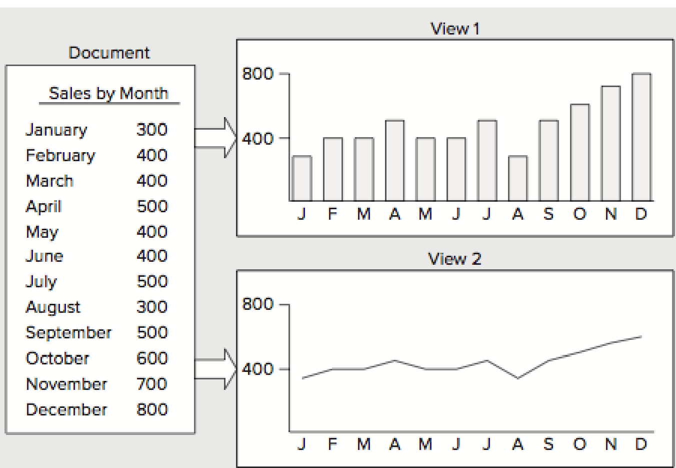


Figure 2.3 : MFC Document/View concept [22].

In EAFE software classes CEAFEDoc, which is a subclass of CDocument class, and CEAFEView, which is a subclass of CView class, created to employ this

document/view structure. The UML class representations of simplified CEAFEDoc and CEAFEView classes are shown in Figure 2.4 with randomly selected data members and member functions.

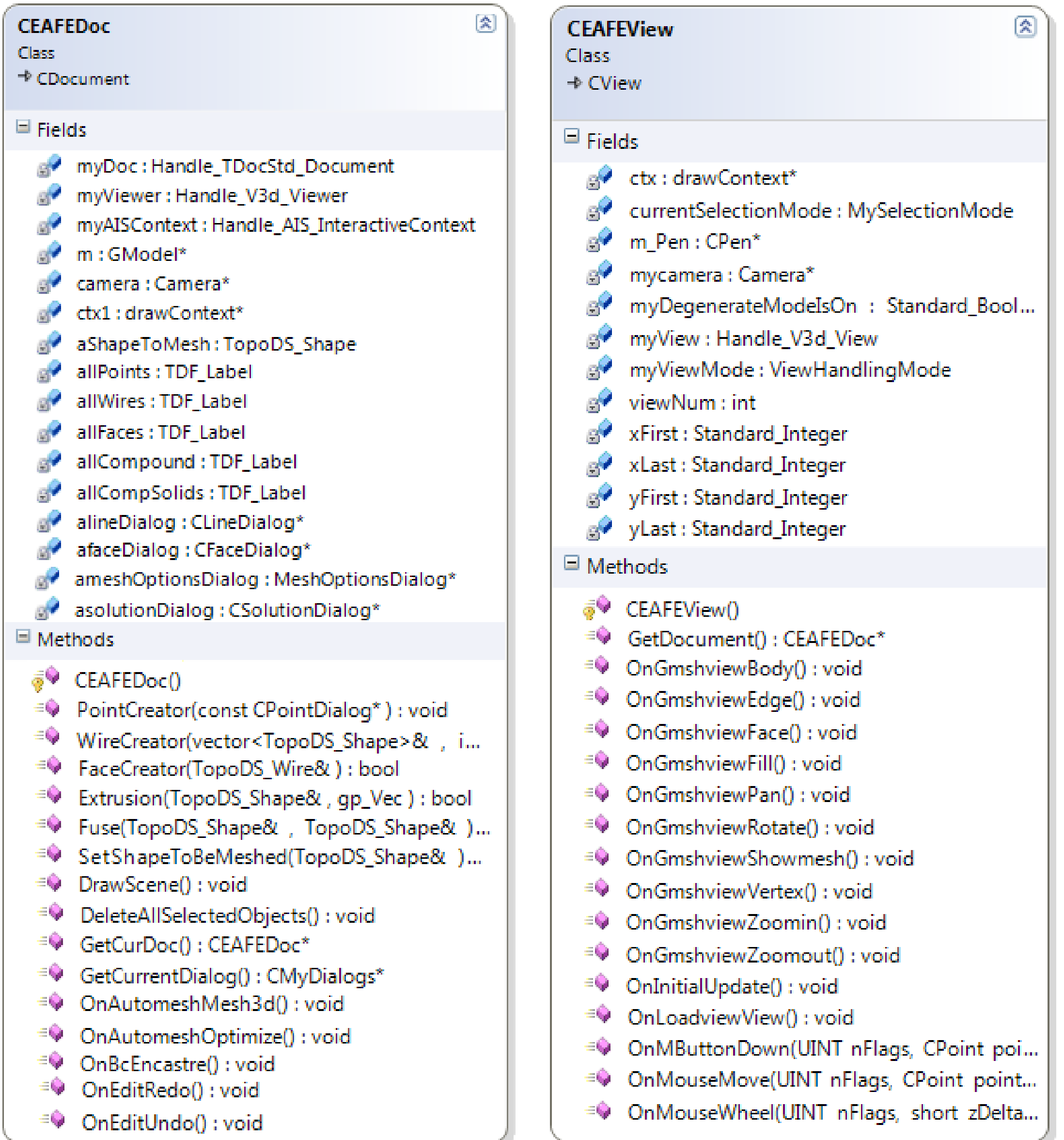


Figure 2.4 : Document/View structure.

3. GEOMETRY MODULE

3.1. Computer Graphics

Computer graphics, as its name suggests, are graphics generated using computers and usually abbreviated as CG. William Fetter, a graphic designer at Boeing Aircraft Co., was coined the term “computer graphics” to describe his job in 1960. But the major breakthrough to demonstrate the potential of computer graphics came when Ivan Sutherland, a Ph.D. student at MIT, developed a drawing program called Sketchpad, as part of his thesis in 1963 [23].

Computer graphics are widespread today. Some of the major industries in which computer graphics are commonly used are entertainment, education, medicine and industrial design. Examples of computer graphics usage in entertainment industry include video games, cartoons, animated films and visual effects. In education area one come across computer graphics in simulations (i.e. flight simulators) and information visualization, while in medicine it is used in medical imaging. In industrial design area, computer graphics are used for computer-aided design and computer-aided manufacturing [24].

The creation, modification, analysis or optimization of an engineering design with the assistance of the computer systems are collectively called computer-aided design [25]. It is argued that the origin of the computer-aided design discipline, same as the most computer graphics fields, is the Sutherland’s Sketchpad program. Companies in the aerospace and automotive industries, realizing the prospects of Sutherland’s revolutionary work, initiated projects to take complete advantage of the computer graphics. The industries wide-ranging interests in the usage of computer graphics in design processes developed and gave birth to multi-billion dollar companies such as Autodesk, Dassault Systems and MSC Software [26].

3.2. 3D Computer Graphics and OpenGL

Computer screen has two dimensions width and height, whereas in real world objects there is one more dimension called depth. The eyes and the brain work together to help us decide the depth of objects. Being supplied with two slightly different images from two eyes, our brains are responsible for combining these images in a way that creates the perception of depth as shown in Figure 3.1 (a). Even though two eyes are needed to truly see in 3D, covering one eye will not cause our 3D perception to disappear abruptly. Because there are other factors, such as perspective, lights, shades, textures and reflections, that can still activate our brain's ability to perceive depth in two dimensions. Artists have long been taking advantage of these factors to depict a three-dimensional scene on a flat canvas, likewise computer graphic designers use the same factors to draw three-dimensional objects on a 2D computer screen.

The use of perspective, which is the way an observer perceives size and details of objects depending on their distance, is the simplest approach to provide an illusion of depth as shown in Figure 3.1 (b). However, inspecting the given figure closely reveals that there is a degree of ambiguity in determining the front and back of the cube. The perspective alone is not enough to accurately represent a three-dimensional object. In addition to the perspective usage, lighting, which refers to the simulation of light, shading, which is using various amount of darkness to illustrate the reflection of light on a surface, texture mapping, which is basically adding a specific pattern or a picture to a surface, and blending, which is mixing different colors to create reflection of a surface on another surface, should be utilized as well [27].

In order to draw a three-dimensional object on a computer screen, a full-scale detailed model has to be constructed first. A model is a mathematical representation of an object on a computer. Models are collection of points that are connected by a number of primitive geometric shapes, such as lines, triangles, curved surfaces, etc. In addition to the geometrical data, models can also include texture, lighting and shading data structures.

Figure 3.2 (a) shows a wireframe model of Utah Teapot, also known as Newell Teapot, which is a 3D model of a regular teapot, Figure 3.2 (c), created by Martin

Newell, a researcher at University of Utah, in 1975, since then extremely frequently used and has become a standard reference object in computer graphics community.

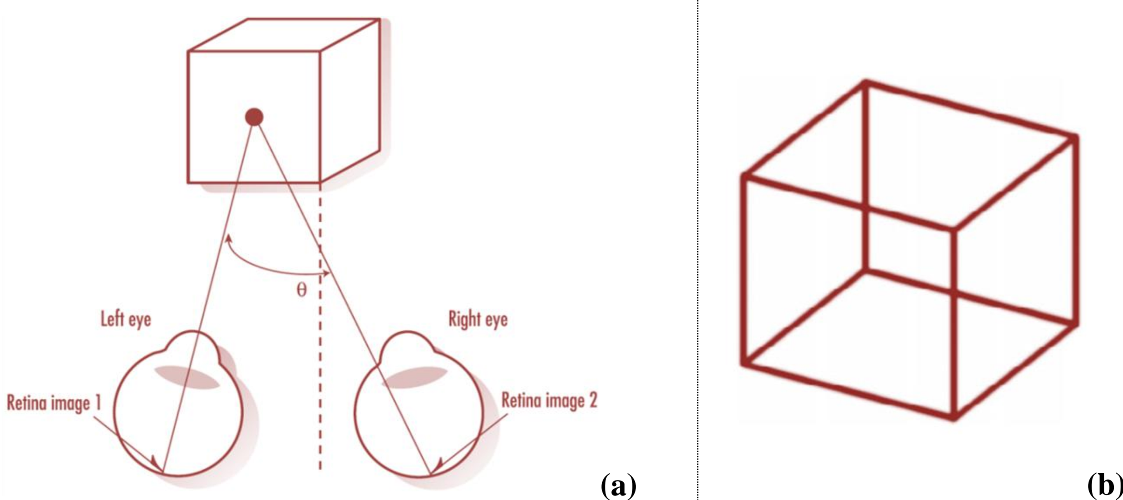


Figure 3.1 : 3D perception: (a)How you see three dimension. (b)A simple wireframe 3D cube.

The process of drawing the final scene on the computer screen from models is called rendering. Historically one of the first complex three-dimensional models to be rendered is the Utah Teapot as mentioned above. Figure 3.2 (b) shows a modern rendered image of Utah Teapot. In computer applications rendering 3D computer graphics are achieved through some specialized application programming interfaces (API). In general, most application programming interfaces are libraries that consist of some object classes, data structures and related variables to accomplish demanding tasks and considerably ease software development procedure. OpenGL (Open Graphics Library) is a language-independent API that can be used in applications to render two-dimensional and three-dimensional computer graphics. The usage of graphics cards to perform 3D graphics operations at high speed is called 3D hardware acceleration. In order to achieve hardware accelerated rendering, OpenGL Interacts with Graphics processing unit(GPU).

Figure 3.2 : Utah Teapot: (a)Wireframe model. (b) A modern render model. (c) Original teapot [28]-[30].

OpenGL API alone is not enough to develop complete applications because it does not have support for opening windows on computer screen or listening mouse and keyboard events. To accomplish these tasks OpenGL must be combined with a general-purpose programming language, such as C++ , Java, Python, etc.

In addition to OpenGL there is another major 3d graphics rendering API called DirectX. However, contrary to OpenGL's cross platform support and open standard, the target operating system(OS) in DirectX API is the Microsoft Windows OS and DirectX API is proprietary. Therefore, OpenGL is more commonly adopted throughout academia and industry and used for widely diverse purposes, from computer-aided design and scientific visualization to entertainment and simulations.

OpenGL API consists of several hundreds of function calls to perform 3D rendering tasks. Using OpenGL's predefined geometric primitives – also called drawing primitives – such as GL_POINTS, GL_LINES, GL_TRIANGLES, GL_QUADS, GL_POLYGON, etc., with these functions, fairly complex objects can be constructed in a Lego-like manner. Figure 3.3 is an example of a simple OpenGL function calls to draw a square with different colored vertices.

The different color values at each vertex are interpolated over the rest of the polygon. Figure 3.4 shows a screenshot of the polygon drawn in a window [31].

In OpenGL each vertex has some quantities called attributes of the vertex. One of these attributes is color as shown above. Another important attribute is the normal vector. Normal vectors are used in lighting calculations. The light beam which comes from a light source, hits a surface and reflects. The properties of this reflection depend on the surface it hits, on the light source and to a great extend on the angle at which the light strikes. OpenGL uses this normal vector which is perpendicular to the surface to calculate the aforementioned angle.

```
glBegin(GL_POLYGON)
    glColor3f( 1.0, 0.0, 0.0 );
    glVertex3f( 20.0, 20.0, 0.0 );
    glColor3f( 0.0, 1.0, 0.0 );
    glVertex3f( 80.0, 20.0, 0.0 );
    glColor3f( 0.0, 0.0, 1.0 );
    glVertex3f( 80.0, 80.0, 0.0 );
    glColor3f( 1.0, 1.0, 0.0 );
    glVertex3f( 20.0, 80.0, 0.0 );
glEnd();
```

Figure 3.3 : An example of a simple OpenGL function.

Figure 3.4 : Screenshot of rendered polygon.

The effect of using different normal vectors on the same geometric shape is illustrated below. In Figure 3.5 (a) normal vectors stored in each vertex are perpendicular to the primitive rectangles and this causes the abrupt change in shading on consequtive rectangles, whereas in Figure 3.5 (b) normal vectors are perpendicular to the curved surface that is being approximated and this causes the smooth change in shading on consequtice rectangles. As it can be inferred, increasing the number of primitive rectangles results in better approximations, in fact, this is exactly what OpenGL does to approximate curved surfaces.

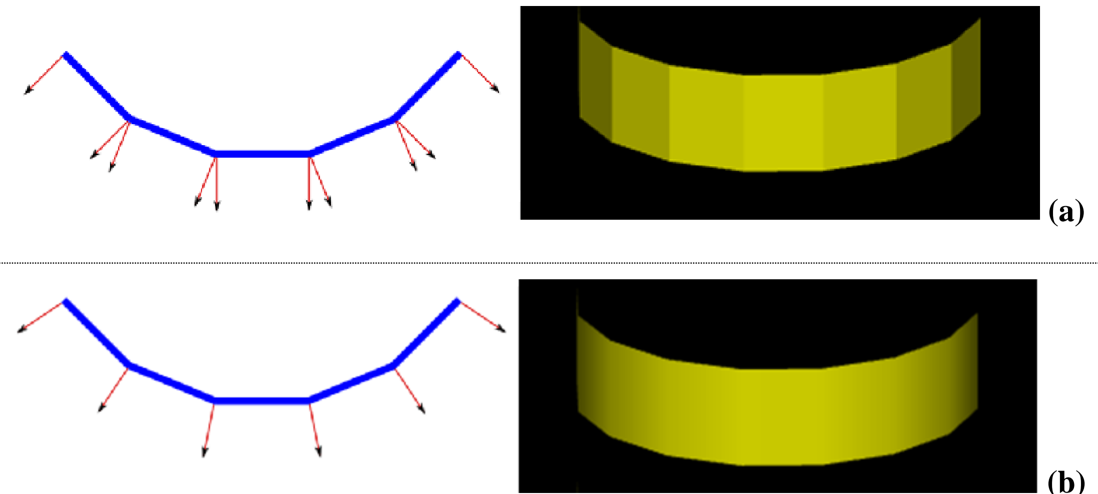


Figure 3.5 : Use of normal vectors in light calculation [32]: (a) Normal vectors perpendicular to rectangles. (b) Normal vectors perpendicular to surface.

Even though some simple shapes could be drawn via supplying normal vectors for each vertex by hand, calculating normal vectors is not an easy task and involve some

non-trivial math [33]. This is one of the reasons why a geometry kernel is needed to build a cad software.

3.3. Geometry Kernel

A Geometry kernel or geometry engine is an independent software component specifically developed to perform 3D solid modeling tasks such as creating, editing, storing, and analyzing 3D models. Many end user applications in computer aided design (CAD), computer aided manufacturing (CAM), and computer aided engineering (CAE) fields are based on geometric modeling kernels. Currently there are two major geometry kernels offered for license: ACIS owned by Spatial and Parasolid owned by Siemens [34]. Table 3.1 shows some well known CAD/CAE softwares and corresponding geometric modeling kernels.

On the other hand, as an alternative to the proprietary softwares Open Cascade S.A.S company offers an open source geometric modeling kernel: Open Cascade Technology (OCCT). OCCT is open source and written in C++ language with an object oriented approach, has adequate documentation and example code fragments, and supports standard geometry file formats such as, ACIS, Parasolid, IGES, STEP, STL, and DXF. Therefore, in the 3D Finite element analysis software developed in this thesis(EAFE), Open Cascade Technology was used as geometric modeling kernel. Taking advantage of some of OCCT's capabilities, a basic solid geometric modeling module was developed and included in the EAFE software.

Table 3.1 : CAD/CAE softwares and related geometry kernels.

CAD/CAE Software	Geometry Kernel
AutoCad	ACIS
SolidWorks	Parasolid
Catia	CGM
Solid Edge	Parasolid
Abaqus	Parasolid
Ansys	Parasolid
MSC.SimXpert	Parasolid

3.4. Open Cascade Technology (OCCT)

Open Cascade Technology is a freely available collection of object-oriented C++ classes designed to assist CAD/CAM/CAE software developers with the rapid development of domain-specific end user applications. OCCT libraries can be divided broadly into four major parts: modelling, visualization, data exchange and application framework.

3.4.1. Modeling module

The 2D and 3D modeling algorithms module brings together a wide range of topological algorithms used in modelling which allow you to model any type of object. Some of the capabilities of modeling module are;

- Creating primitives such as prism, cylinder, cone and torus.
- Performing boolean operations (addition, subtraction and intersection)
- Tweaking constructions using fillets, chamfers and drafts.
- Modeling constructions using offsets, shelling, hollowing, and sweeps.
- Computing properties such as surface, volume, center of gravity, curvature.
- Computing geometry using projection, interpolation, approximation

3.4.2. Visualization module

Includes services that allow you to manage object display and manipulate views. Some of the capabilities of visulization module are:

- 3D Rotation, zooming and panning,
- Shading.

3.4.3. Data exchange module

Provides import and export functions of OCCT models to and from standard formats such as IGES and STEP.

3.4.4. Application framework module

- Association between non-geometrical application data and geometry.
- Parameterization of models.

3.5. Implementation of OCCT Modules in EAFE Software

In the development of EAFE 3D Finite element analysis software, OCCT modules are used to provide CAD functionality in the software.

Partial implementation of modeling and visualization components of OCCT in the developed EAFE software's geometry module is shown in Figure 3.6. The first highlighted section includes necessary buttons such as Fill, Pan, Rotate, Zoom, etc., to manipulate views. Each of these buttons has a unique event handler in EAFEView class, which calls related functions with required parameters from OCCT's visualization module to adjust the view as desired.

The second highlighted section displays undo, redo and delete buttons on the edit panel. These buttons are one of the most important features of modern softwares, because they provide recovery from mistakes. This functionality is included in EAFE software with the use of OpenCascade's Application Framework(OCAF) module which is going to be explained in the subsequent pages.

The third highlighted section shows three panels named: geometry, solid, and modeling. Geometry panel includes some buttons for primitive geometric shapes that can be used as starting points for solid modeling. Solid panel contains buttons to create some frequently used solid shapes such as boxes, spheres, and cylinders. Lastly, modeling panel has a number of buttons that are useful to make desired shapes. Event handlers, in which OCCT's modeling module functions are called, for these buttons are implemented in EAFEDoc class.

Exchanging data is another fundamental feature that most CAD softwares has in common. The fourth highlighted section in Figure 3.7 shows EAFE software's import and export functionality which is provided by OCCT's data exchange module and can be used to import or export solid models to or from other well known CAD softwares by using standard formats such as STEP or IGES.

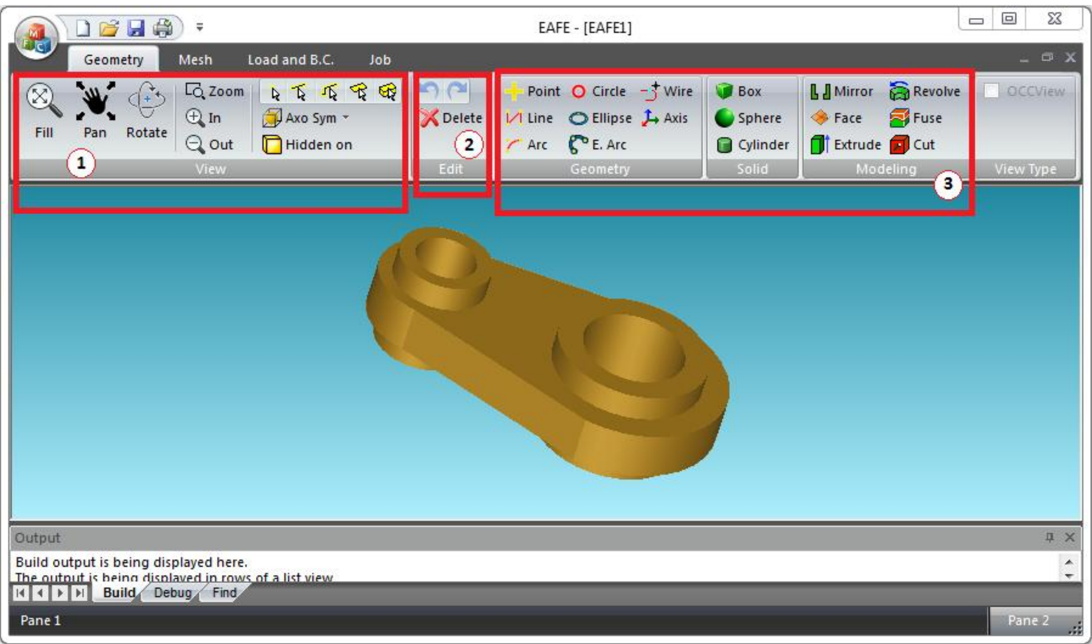


Figure 3.6 : Geometry module of EAFE software.

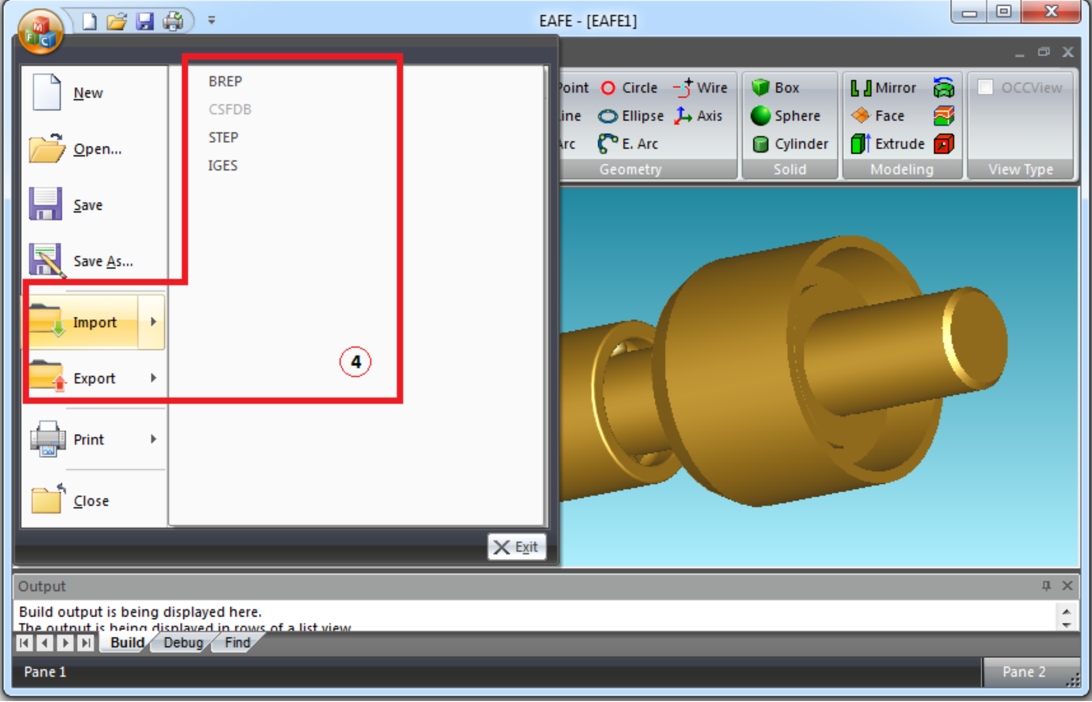


Figure 3.7 : Import export properties of EAFE software.

3.6. Open Cascade Application Framework (OCAF)

One of the most essential modules of OCCT is the Open Cascade Application Framework (OCAF). OCAF is much more than just one toolkit among many in OCCT libraries. Since it can handle any data and algorithms in these libraries –

modeling algorithms, topology or geometry – OCAF is a logical supplement to these libraries and is going to be explained here in detail.

OCAF is a rapid application development (RAD) framework used for specifying and organizing application data. To do this, OCAF provides:

- Ready-to-use data common to most CAD/CAM applications,
- A scalable extension protocol for implementing new application specific data,
- An infrastructure
  - To attach any data to any topological element
  - To link data produced by different applications
  - To register the modeling process – the creation history, or parametrics, used to carry out modifications.

Using OCAF, the application designer concentrates on the functionality and its specific algorithms. In this way, he avoids architectural problems notably implementing undo-redo and saving application data. In OCAF, all of the above are already handled for the application designer, allowing him to reach a significant increase in productivity.

In OCAF, data structure is reference key-driven. The reference key is implemented in the form of labels. Application data is attached to these labels as attributes. By means of these labels and a tree structure they are organized in, the reference key aggregates all user data, not just shapes and their geometry. These attributes have similar importance; no attribute is master in respect of the others [35].

The reference keys of a model - in the form of labels - have to be kept together in a single container. This container is called a document. OCAF documents are in turn managed by an OCAF application. Inside a document, there is a data framework. This is a set of labels organized in a tree structure. Figure 3.8 shows a rudimentary example of an OCAF data framework, in which the the tags are illustrated in the circles, and the labels are illustrated under the circles as tag lists.

The data framework offers a single environment in which data from different application components can be handled. This allows you to exchange and modify

data simply, consistently, with a maximum level of information, and with stable semantics. The building blocks of this approach are: 1. Tag, 2. Label, 3.Attribute.

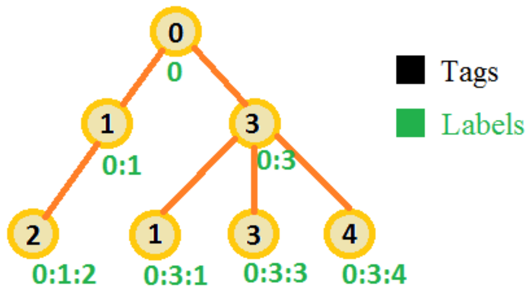


Figure 3.8 : A basic OCAF data framework.

The first label in a framework is the root label of the tree. Each label has a tag expressed as an integer value, and a label is uniquely defined by an entry expressed as a list of tags from the root, 0:3:1, for example. Each label can have a list of attributes, which contain data, and several attributes can be attached to a label.

The data framework in EAFE Software's geometry module is OCAF based and shown in Figure 3.9. The simplified tree like structure holds tags, labels and the label's attributes as shown in the figure. Each geometric primitive or solid part created in the application is an attribute and stored under this tree structure with associated label. The labels can have a list of attributes including name, number, color, etc., along with the shape. When a change, modification or removal is needed for any shape the particular label for this shape can be used to retrieve the shape. A label's entry is its persistent address in the data framework.

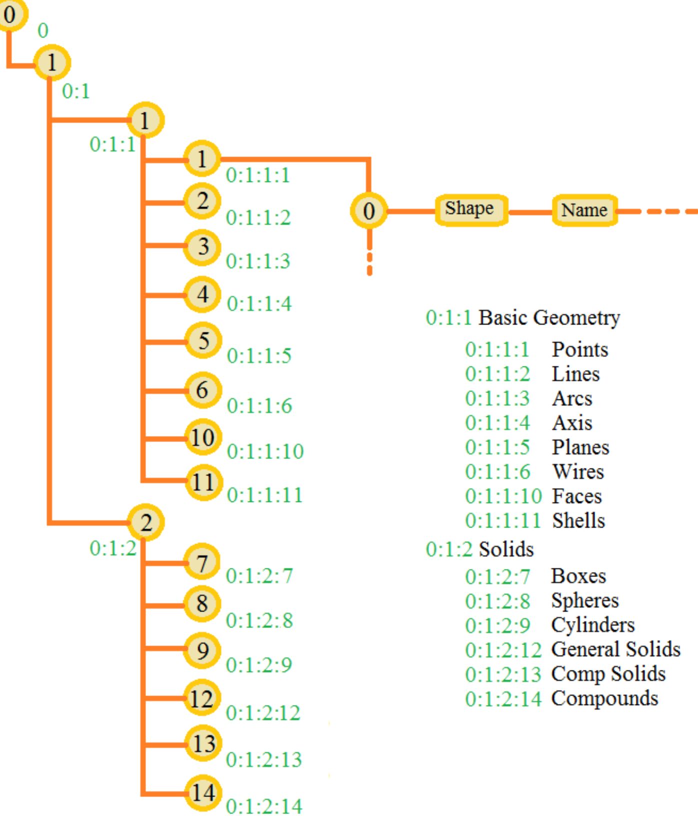


Figure 3.9 : EAFE software's OCAF based data framework.

4. MESH MODULE

4.1. Mesh Generators

One of the most important and time consuming steps in the finite element analysis is mesh generation. A mesh is a geometrical discretization of a computational domain. During this discretization the continuous domain split into geometrically simple and smaller subdomains called elements. Examples of elements used in finite element analysis include lines, triangles and quadrangles in 2D and tetrahedrons, hexahedrons, prisms and pyramids in 3D.

An unstructured (or irregular) mesh is a tessellation of a domain by simple shapes in an irregular pattern. The process of obtaining an appropriate mesh is called mesh generation. In unstructured mesh generation, triangular and tetrahedral elements are by far the most common used element types. In general, Octree, Delaunay and Advancing front techniques are applied for unstructured mesh generation. There are public domain and commercial mesh generators which are distributed by software vendors, research labs and educational institutions. Two of the prominent open source mesh generators which also offers built-in post processing facilities are Salome and Gmsh.

Salome is an open source software that provides a generic platform for pre-processing and post-processing for numerical simulation. It is based on an open and flexible architecture made of reusable components [36]. Similarly, Gmsh is a three-dimensional finite element mesh generator with a build-in geometry engine and post-processor. It aims to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities [37].

As it is the case in the most open source softwares both Salome and Gmsh libraries do not have enough documentation to describe member functions. Contrary to Salome, Gmsh supports 64bit Windows OS and is written in C++, it is light and easy to use, therefore, Gmsh is used as a mesh framework in the EAFE Software.

4.2. GMSH Mesh Framework

GMSH is an open source mesh generator developed and is being maintained by Christophe Geuzaine from University of Liège and Jean-François Remacle from Université catholique de Louvain, in order to meet the expectations of researchers and engineers in academia and industry. Making use of lines, triangles and tetrahedrons, Gmsh generates 1D, 2D and 3D finite element meshes with adjustable element size. Gmsh also provides a post-processor that can load and manipulate scalar, vector and tensor maps. Gmsh is powerful enough to be used in academic and engineering applications.

Gmsh uses four model entities to represent 3D solids: vertices, edges, faces and regions. The logic behind this representation is that any solid can be defined as a volumetric region bounded by a set of surfaces, surfaces bounded by a sequence of edges and edges bounded by two vertices at each end. Taking into account this geometric representation, the discretization process in Gmsh is designed to go from bottom to up following three main steps as shown below:

- The first discretized entities are edges,
- Using discretized edges, surfaces are triangulated,
- Making use of surface mesh data, volumetric regions are tetrahedralized.

Gmsh has three different algorithm options for 2D meshing: Mesh adapt, Delaunay, and Frontal. Delaunay and Frontal algorithms are standard algorithms. In addition to these algorithms Gmsh offers a new surface meshing technique in which the notion of local mesh modifications are used. In the MeshAdapt algorithm discretized domain is locally modified such that an edge is split if it is too long or is collapsed if it is too short, and edges are swapped if swapping an edge results in a better geometric configuration. Gmsh uses Delaunay and Frontal algorithms for 3D unstructured discretization [38].

4.3. Implementation of Gmsh in EAFE

In order to provide meshing ability some of the Gmsh library's capabilities such as 1D, 2D and 3D mesh generation, mesh optimization, mesh size manipulation and increasing mesh density in critical regions of the discretized shape, are used in EAFE

Software. Along with these mentioned capabilities, some of the wide range of options that Gmsh library offers to control the behavior of mesh commands, and the way meshes are displayed, are provided in the EAFE Software.

Implementation of Gmsh through mesh related buttons in EAFE software's Mesh module is highlighted in Figure 4.1. Similar to the buttons in Geometry module, each of these buttons also has event handlers in EAFEDoc class which is one of the two main classes in EAFE software. Pressing one of these buttons results in a call to the corresponding event handler function in which Gmsh library's related function employed.

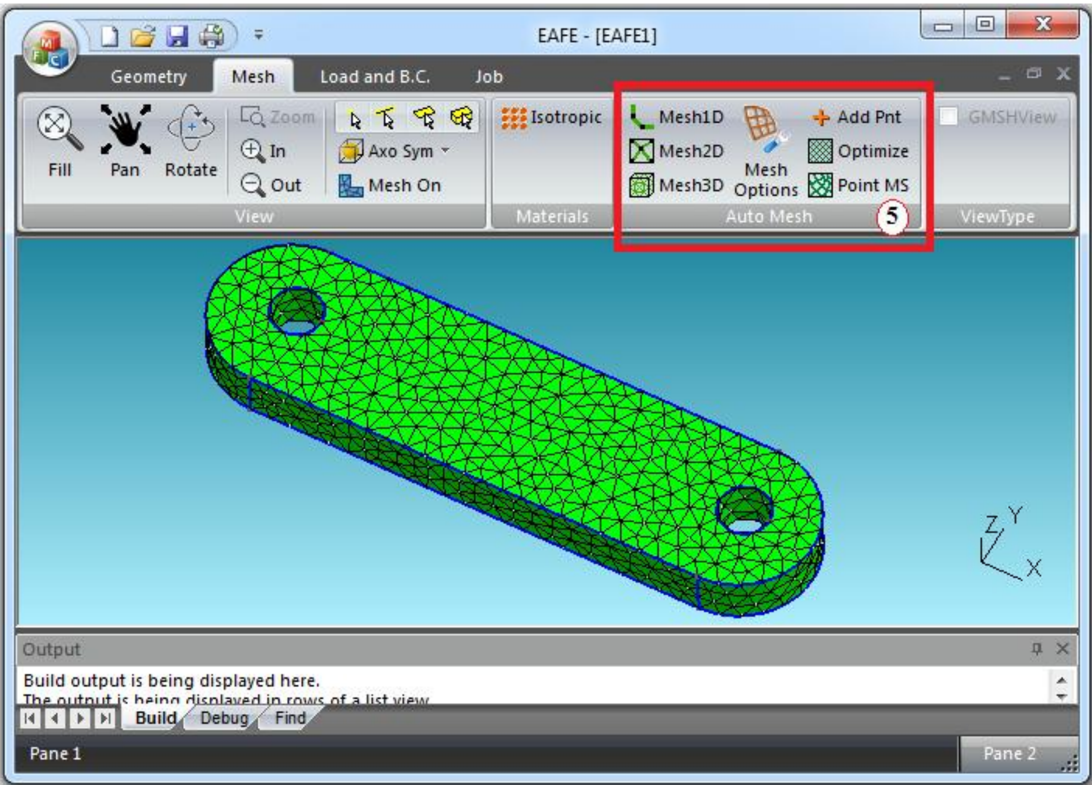


Figure 4.1 : EAFE software with GMSH mesh framework.

Overly simplified code fragments taken from original EAFE source code and put together to illustrate the use of Gmsh library is given in Figure 4.2. Gmsh must be initialized before using its accompanying functions, hence statement in line 1 initializes Gmsh. Line 2 creates a geometric model with GModel class which is one of the most important classes in Gmsh library. Line 4 imports an Open Cascade shape that is created in advance with the EAFE's geometry module, into the geometric model and checks whether any problem occur. In case of a problem it prompts an error message in a dialog box.

```
1.  GmshInitialize ();      // initialize gmsh
2.  GModel* myModel = new GModel();
3.  // import open cascade shape into the geometric model
4.  if (!(myModel->importOCCShape((void*)&aShapeToMesh)))
5.  {
6.     AfxMessageBox( L"Error during shape loading !" );
7.     return;
8.  }
9.  // generating a 3d mesh
10.  try {
11.     myModel ->mesh(3);
12.  } catch (...) {
13.     AfxMessageBox(L"Error in gmsh--aborting mesh!\n");
14.  }
15.  // update View and save the mesh data
16.  DrawScene();
17.  myModel->writeMSH("C:/Users/EafeTemp/Mesh/part.msh");
18.  // delete the geometric model and terminate gmsh
19.  delete myModel;
20.  GmshFinalize();
```

Figure 4.2 : Gmsh library usage.

An exception is a problem that rarely occurs during a program's execution. Try and catch blocks are used to handle exceptions in programs. Using exception handling enables programmers to develop fault tolerant programs. Statements from line 10 to 14 generate a 3D mesh for the geometric model using a try and catch block to handle any kind of exception that could happen during discretization process. Line 16 updates the scene and line 17 writes mesh data to the file in the given directory. Line 19 deletes the model and additional entities such as geometry data, mesh data, etc. Line 20 terminates the process.

Another important Gmsh functionality that is used in EAFE software is the mesh size manipulation. The effect of changing element size at certain points over the shape to the mesh density is shown in Figure 4.3. An Open Cascade solid generated in EAFE geometry module is given in Figure 4.3 (a). The solid shape imported to the EAFE mesh module and discretized with constant mesh size as shown in Figure 4.3 (b). By means of changing element size at two vertices a new mesh generated to demonstrate the element size manipulation capability of the software and shown in Figure 4.3 (c).

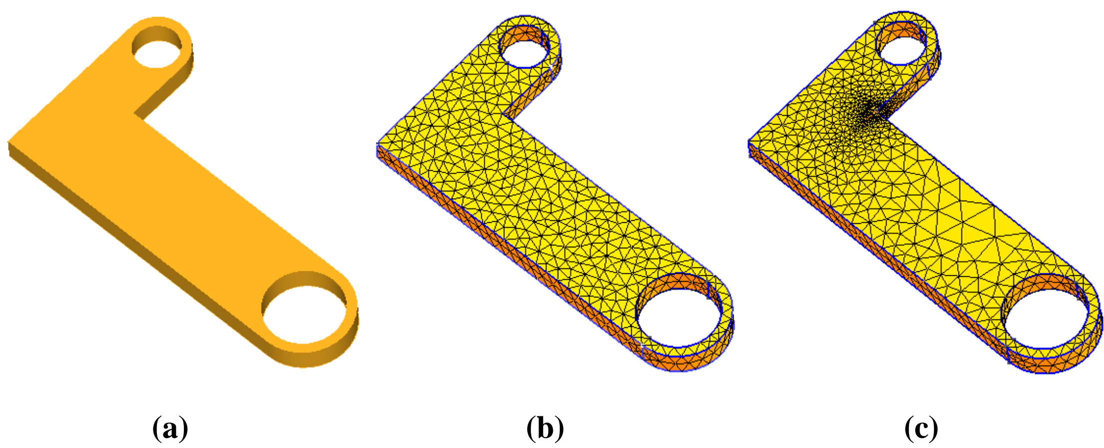


Figure 4.3 : A solid shape with different mesh options: (a) Solid shape. (b) Constant mesh size. (c) Variable mesh size.

5. SOLVER MODULE

5.1. Object-Oriented Programming

In contrast to the procedural programming's function oriented approach, in the object-oriented programming the application is build around its data, which is stored in small packages called objects. A brief introduction to the object-oriented programming philosophy is provided in Chapter 2. Here the fundamental concepts in object-oriented programming such as object, class, method, inheritance, polymorphism, etc., are introduced.

5.1.1. Fundamental concepts in object-oriented programming

Moving from procedural programming to object-oriented programming is not an easy task. To successfully develop a software with object-oriented programming approach, a developer must have a thorough understanding of the essential concepts given in this section.

5.1.1.1. Object

Almost each programming language has standard data types such as integer, float, double, string, etc. In essence, an object is a new type of data variable that is defined by the user and anything can be an object. Objects have data members composed of classic variables and/or other new user defined variables. Objects also have member functions that view or manipulate its encapsulated data members.

5.1.1.2. Class

Classes are the building blocks of object-oriented programming. A software developed with object oriented approach is basically a collection of classes that communicate with each other via messages to complete the required tasks. Classes are user defined data types that are used to instantiate objects as shown in Figure 5.1. In this figure ahmet and mehmet objects are instantiated from Person class and as many objects as needed can be initiated from a class.

```
int a = 12;             // a is a classic data variable an integer
double b = 2.3;         // b is a classic data variable a double
bool stop = false;      // stop is a classic data variable a boolean

Person ahmet;           // ahmet is a new data variable a Person
Person mehmet;          // mehmet is a new data variable a Person
```

Figure 5.1 : A simple code fragment to show classes and objects.

To illustrate the class concept the UML representation of this person class with possible member fields like age, gender, weight and member methods like constructer, get and set functions are given in Figure 5.2.

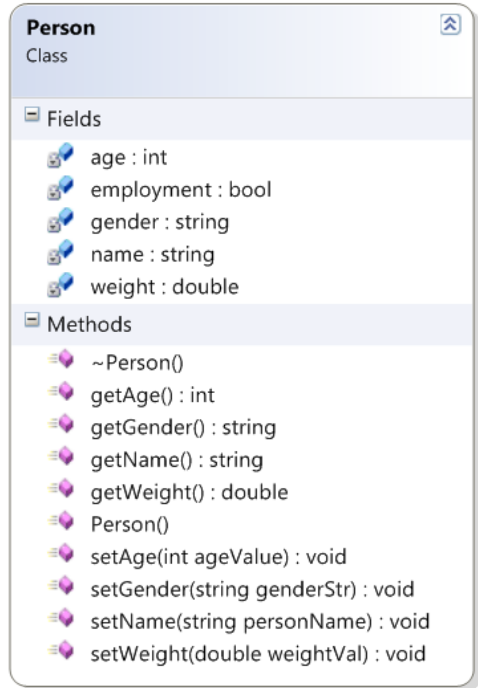


Figure 5.2 : An example class structure.

5.1.1.3. Encapsulation

Encapsulation is a mechanism in which only a classes methods are allowed to access the same classes fields. For example, in the given person class the fields such as age, employment, weight, etc., are only accessible to the methods in the person class. If another class needs to view or modify a field in the person class, due to its restricted access, it must use the related get or set method provided in the person class. Encapsulation is an important topic in object-oriented programming, because of the fact that it provides softwares flexibility and modularity.

5.1.1.4. Method

A method in object-oriented programming is similar to a procedure, function, or routine in procedural programming languages. The main distinction is that, methods are always associated with classes.

In object-oriented programming methods are used to view and manipulate data members (attributes) of objects. A simple example is given in Figure 5.3 to show method usage in object-oriented programming. In the figure age and gender attributes for ahmet and mehmet objects are set by using setAge and setGender methods respectively. Using methods to set an object's attributes, can be very useful to prevent assigning invalid values to the attributes. For example, inside the setAge and setGender methods' implementations, it is possible to check age and gender parameters supplied in the method calls, before setting the object's age and gender attributes. This is known as keeping the object in a consistent state.

```
ahmet.setAge( 25 );  // sets age to 25 for ahmet object
mehmet.setAge( 35 ); // sets age to 35 for mehmet object

ahmet.setGender( "Male" );              // sets gender for ahmet object
mehmet.setGender( ahmet.getGender() ); // sets gender for mehmet object

ahmet.setAge(-10); // in setAge method's implementation negative values
                   // are set to 0 to maintain object's integrity
```

Figure 5.3 : A simple code fragment to show method usage.

5.1.1.5. Inheritance

Inheritance is one of the essential features of object-oriented programming. It is a mechanism that facilitates software reuse. By using inheritance a new class, called derived class, can be built on a pre-existing class, called base class in C++ language. In general, derived classes inherits base classes fields and methods, and adds its particular variables and methods. Therefore, derived classes are more specific than their base classes. The use of inheritance makes it possible to organize objects into a hierarchy, and define relationships with each other.

A simple example is given to illustrate the use of inheritance to create new classes in Figure 5.4. In this example Student and Employee classes both inherit Person classes fields and methods. Similarly, Undergraduate and Graduate classes adds their own specific variables and methods on the inherited fields and methods from the Student class. This example shows how a base class functionality is extended by derived

classes. Creating new classes in this way produces neat and clean code that is reusable and easier to understand.

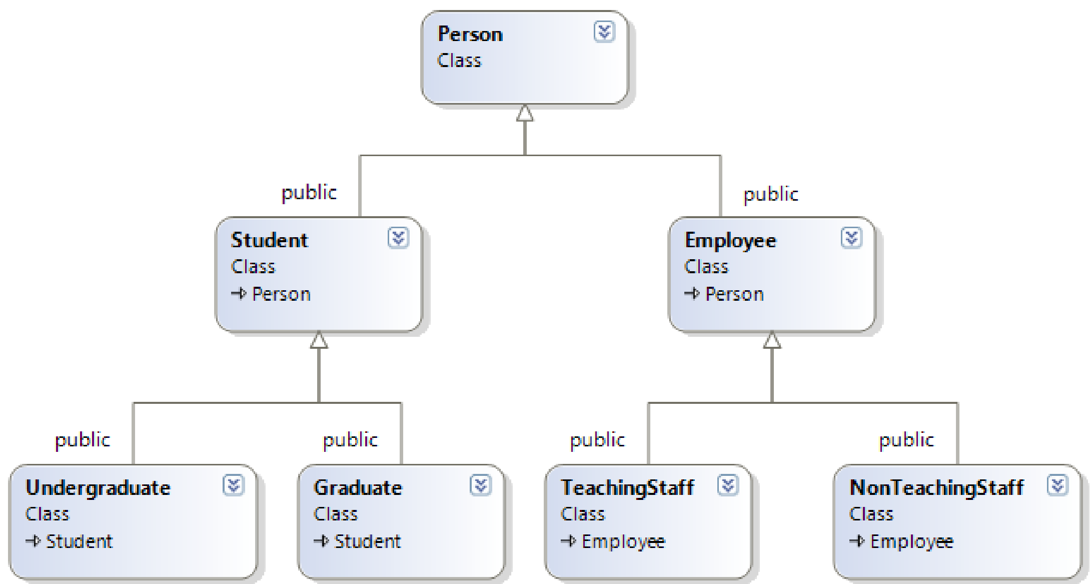


Figure 5.4 : An inheritance hierarchy example.

5.1.1.6. Polymorphism

Polymorphism is the ability of a field, method or object to take on multiple forms. Using polymorphism, a developer is able to program in the general rather than program in the specific. In polymorphism, sending the same message to different objects can bring about different behaviour, depending on the object type, and the exact behaviour is determined at program execution time. Using this property of polymorphism, messages can be sent to the objects, without knowing the types of the objects. This is an important property of polymorphism which makes designing and implementing easily extensible object-oriented systems possible.

5.2. Finite Element Method

Differential or integral equations are used to describe many physical phenomena in engineering such as elasticity, heat, sound, or fluid flow. In complex field problems numerical solution methods are used, because solution of these equations with classic analytical approaches becomes virtually impossible. The finite element method (FEM) is a numerical approach to find approximate solutions to the differential and integral equations [39].

The finite element method consists of three steps: 1. Preprocessing, 2. Processing (Solution), 3. Postprocessing.

In the preprocessing step problem domain is discretized, material properties and boundary conditions are defined, and loads are applied. In the processing step mesh data from the previous step is used to create element matrices. In the case of distributed loads such as body forces and surface loads, the equivalent nodal load vectors are obtained for these elements. Subsequently, element matrices and load vectors are used to create global matrices such as stiffness matrix [K], mass matrix [M], and load vector {F}. The boundary conditions are applied on the matrix equation such as [K]{Q} = {F} for the static analysis of structures. Here {Q} is the vector of unknown displacements. Finally, the results are graphically displayed in the postprocessing step.

5.2.1. Three-dimensional stress analysis

Elasticity is an important subject that deals with determination of the stress, strain, and displacement distribution in an elastic solid under the influence of external forces. Following the usual assumptions of linear, small-deformation theory, the formulation establishes a mathematical model that allows solutions to problems that have applications in many engineering and scientific fields. Applications in aeronautical and aerospace engineering include stress, fracture, and fatigue analysis in aerostructures.

The basic aim of structural mechanics problem is to determine the distribution of displacements and stresses under the loading and boundary conditions. A mathematical model of the structural problem is necessary to find the desired distributions by using FEM. An understanding of all the basic equations of structural mechanics is essential to devise an appropriate or adequate mathematical model. Hence, the basic equations of solid mechanics are summarized in the following sections for ready reference in the formulation of FE equations [40].

5.2.1.1. Fundamental equations

If the deformation of an elastic body is considered under the applied external forces, any point of the body is displaced from a point to another point. A displacement vector can be defined for any point of the body, and it can be resolved into three

displacement components u, v and w in the x, y and z axis, repectively. Displacements are unkown functions of coordinates.

On the other hand six independent strain components and corresponding six stress components can be defined for any point of the body. Three of them are normal strains or stresses and three others shear strains and stresses. The displacements, strains and strains are unknowns of an elasticity problem. Three-dimensional stresses on an infinitesimal element are given in Figure 5.5.

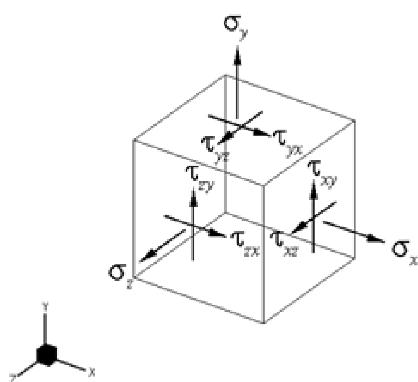


Figure 5.5 : Three dimensional stresses on an element [41].

The stresses and strains are given by

$$\{\sigma\} = [\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{yz}, \tau_{zx}]^T \quad \textbf{(5.1)}$$

$$\{\varepsilon\} = [\varepsilon_x, \varepsilon_y, \varepsilon_z, \gamma_{xy}, \gamma_{yz}, \gamma_{zx}]^T \quad \textbf{(5.2)}$$

The stress-strain relations are given by Hooke's law as

$$\{\sigma\} = [E]\{\varepsilon\} \quad \textbf{(5.3)}$$

where the constitutive matrix $[E]$ in equation **(5.3)** for an isotropic material is given by

$$[E] = \frac{E}{(1+v)(1-2v)}\begin{bmatrix} 1-v & v & v & 0 & 0 & 0 \\ & 1-v & v & 0 & 0 & 0 \\ & & 1-v & 0 & 0 & 0 \\ & & & \frac{1-2v}{2} & 0 & 0 \\ & Symmetric & & & \frac{1-2v}{2} & 0 \\ & & & & & \frac{1-2v}{2} \end{bmatrix} \quad \textbf{(5.4)}$$

The displacements in x, y and z directions are represented in a vector as

$$\{\delta\} = [u, v, w] \tag{5.5}$$

For the three-dimensional case, the strain-displacement relations can be written as follows

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{5.6}$$

or using **(5.2)** and **(5.5)** the equation in **(5.6)** can be shown in matrix form as

$$\{\varepsilon\} = [d]\{\delta\} \tag{5.7}$$

where $[d]$ is called strain-displacement operator.

5.2.1.2. Tetrahedral element (Tet-4)

The four node tetrahedral element is generally abberivated in computer programs as Tet-4, and because of its linear shape functions, it is also called the linear tetrahedron. The tetrahedral element is the simplest solid element [42]. Element formulations for tetrahedral element are developed in this section.

Each node in tetrahedral element has three degrees of freedom and the vector of the nodal degrees of freedom is given by

$$\{q\} = [u_1, v_1, w_1, u_2, v_2, w_2, u_3, v_3, w_3, u_4, v_4, w_4]^T \tag{5.8}$$

The displacements u, v, and w at any point in the element can be found by interpolating displacement values at four nodes, and this relationship is given by

$$\{\delta\} = [N]\{q\} \tag{5.9}$$

where $[N]$ matrix is given by

$$[N] = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & 0 & 0 & N_3 & 0 & 0 & N_4 \end{bmatrix} \quad \textbf{(5.10)}$$

The N_i shape functions are defined by using the master element given in Figure 5.6 as follows

$$N_1 = \xi \quad N_2 = \eta \quad N_3 = \zeta \quad N_4 = 1 - \xi - \eta - \zeta \quad \textbf{(5.11)}$$

In these functions $N_i = 1$ at node i, and $N_i = 0$ at all other nodes [43].

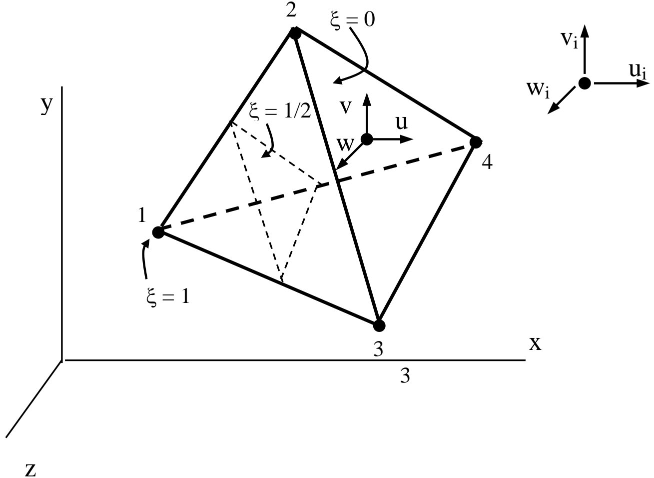


Figure 5.6 : Master element used in shape functions.

The same shape functions can be used to define x, y, z coordinates of any point at which u, v, w displacements are interpolated. The isoparametric transformation is given by

$$x = N_1 x_1 + N_2 x_2 + N_3 x_3 + N_4 x_4$$

$$y = N_1 y_1 + N_2 y_2 + N_3 y_3 + N_4 y_4 \quad \textbf{(5.12)}$$

$$z = N_1 z_1 + N_2 z_2 + N_3 z_3 + N_4 z_4$$

Using **(5.11)**, the equations in the **(5.12)** can be rewritten as

$$x = x_4 + x_{14}\xi + x_{24}\eta + x_{34}\zeta$$

$$y = y_4 + y_{14}\xi + y_{24}\eta + y_{34}\zeta \quad \textbf{(5.13)}$$

$$z = z_4 + z_{14}\xi + z_{24}\eta + z_{34}\zeta$$

where the notations x_{ij}, y_{ij}, and z_{ij} are given by

$$x_{ij} = x_i - x_j \quad y_{ij} = y_i - y_j \quad z_{ij} = z_i - z_j \tag{5.14}$$

Using equations **(5.7)** and **(5.9)**, the following equation can be written

$$\{\varepsilon\} = [B]\{q\} \tag{5.15}$$

where $[B]$ matrix is equal to $[d][N]$ and is given by

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \frac{\partial N_3}{\partial x} & 0 & 0 & \frac{\partial N_4}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & 0 & \frac{\partial N_3}{\partial y} & 0 & 0 & \frac{\partial N_4}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & 0 & 0 & \frac{\partial N_3}{\partial z} & 0 & 0 & \frac{\partial N_4}{\partial z} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial z} & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_4}{\partial z} & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial z} & 0 & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial z} & 0 & \frac{\partial N_4}{\partial x} \end{bmatrix} \tag{5.16}$$

Using the chain rule, the relation between the derivatives of the shape functions N_i with respect to ξ, η, ζ and cartesian derivatives are given by

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} = [\,J\,] \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} \quad i = 1,2,3,4 \tag{5.17}$$

where the Jacobian matrix $[\,J\,]$ is given by

$$[\,J\,] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} x_{14} & y_{14} & z_{14} \\ x_{24} & y_{24} & z_{24} \\ x_{34} & y_{34} & z_{34} \end{bmatrix} \tag{5.18}$$

Using the inverse of the Jacobian matrix, the equation in **(5.17)** can be written as

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial \mathrm{z}} \end{Bmatrix} = [\,\mathrm{J}\,]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \\ \frac{\partial N_i}{\partial \zeta} \end{Bmatrix} \quad i = 1,2,3,4 \tag{5.19}$$

where the inverse of the Jacobian matrix is given by

$$[A] = [\,\mathrm{J}\,]^{-1} = \frac{1}{|\,\mathrm{J}\,|} \begin{bmatrix} y_{24}z_{34} - y_{34}z_{24} & y_{34}z_{14} - y_{14}z_{34} & y_{14}z_{24} - y_{24}z_{14} \\ x_{34}z_{24} - x_{24}z_{34} & x_{14}z_{34} - x_{34}z_{14} & x_{24}z_{14} - x_{14}z_{24} \\ x_{24}y_{34} - x_{34}y_{24} & x_{34}y_{14} - x_{14}y_{34} & x_{14}y_{24} - x_{24}y_{14} \end{bmatrix} \tag{5.20}$$

and the determinant of the Jacobian matrix $|\,\mathrm{J}\,|$ is given by

$$|\,\mathrm{J}\,| = x_{14}(y_{24}z_{34} - y_{34}z_{24}) + y_{14}(z_{24}x_{34} - z_{34}x_{24}) + z_{14}(x_{24}y_{34} - x_{34}y_{24}) \tag{5.21}$$

Using equations **(5.19)** and **(5.20)** the $[B]$ is modified as

$$[B] = \begin{bmatrix} A_{11} & 0 & 0 & A_{12} & 0 & 0 & A_{13} & 0 & 0 & -\tilde{A}_1 & 0 & 0 \\ 0 & A_{21} & 0 & 0 & A_{22} & 0 & 0 & A_{23} & 0 & 0 & -\tilde{A}_2 & 0 \\ 0 & 0 & A_{31} & 0 & 0 & A_{32} & 0 & 0 & A_{33} & 0 & 0 & -\tilde{A}_3 \\ 0 & A_{31} & A_{21} & 0 & A_{32} & A_{22} & 0 & A_{33} & A_{23} & 0 & -\tilde{A}_3 & -\tilde{A}_2 \\ A_{31} & 0 & A_{11} & A_{32} & 0 & A_{12} & A_{33} & 0 & A_{13} & -\tilde{A}_3 & 0 & -\tilde{A}_1 \\ A_{21} & A_{11} & 0 & A_{22} & A_{12} & 0 & A_{23} & A_{13} & 0 & -\tilde{A}_2 & -\tilde{A}_1 & 0 \end{bmatrix} \tag{5.22}$$

where $\tilde{A}_1$, $\tilde{A}_2$, and $\tilde{A}_3$ are

$$\tilde{A}_1 = A_{11} + A_{12} + A_{13} \qquad \tilde{A}_2 = A_{21} + A_{22} + A_{23} \qquad \tilde{A}_3 = A_{31} + A_{32} + A_{33} \tag{5.23}$$

Element stiffness matrix

Element stiffness matrix is given by

$$[k] = \iiint_V [B]^T [E][B] dV \tag{5.24}$$

If the element is a four node tetrahedral element the $[E]$ and $[B]$ matrices are constant, therefore, equation **(5.24)** simplifies to

$$[k] = V[B]^T[E][B] \tag{5.25}$$

where V is the volume of the element given by

$$V = \frac{1}{6}|\det J| \quad \textbf{(5.26)}$$

Force terms

The potential term associated with body force is

$$\int_e \{u\}^T\{f\}dV = \{q\}^T \iiint [N]^T\{f\}detJd\xi d\eta d\zeta = \{q\}^T\{f^e\} \quad \textbf{(5.27)}$$

If $f_x, f_y, and\ f_z$ components of the {f} load are constant, solution of the integral in equation **(5.27)** gives the element body force vector by

$$\{f^e\} = \frac{V_e}{4}\left[f_x, f_y, f_z, f_x, f_y, f_z, f_x, f_y, f_z, f_x, f_y, f_z\right]^T \quad \textbf{(5.28)}$$

The potential term associated with surface traction is

$$\int_{A_e} \{u\}^T\{T\}dA = \{q\}^T \int_{A_e} [N]^T\{T\}dA = \{q\}^T\{T^e\} \quad \textbf{(5.29)}$$

If $T_x, T_y, and\ T_z$ components of the {T} load are constant, solution of the integral in equation **(5.29)** gives the element traction load vector by

$$\{T^e\} = \frac{A_e}{3}\left[T_x, T_y, T_z, T_x, T_y, T_z, T_x, T_y, T_z, 0, 0, 0\right] \quad \textbf{(5.30)}$$

The potential energy of an element can be written as

$$\Pi^e = U^e + W^e = \frac{1}{2}\{q\}^T[k]\{q\} - \{q\}^T\{f^e\} - \{q\}^T\{T^e\} \quad \textbf{(5.31)}$$

Assembly procedure

Total potential energy for the structure can be obtained by summing potential energies of individual elements

$$\Pi = \sum_{e=1}^{E} \Pi^e - \{Q\}^T\{F_c\} \quad \textbf{(5.32)}$$

where {Q} is the nodal displacement vector as shown below

$$\{Q\} = \begin{Bmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_N \end{Bmatrix} = \sum_{e=1}^{E} \{q\} \quad \textbf{(5.33)}$$

Thus, the total potential energy is given by

$$\begin{aligned} \Pi &= \frac{1}{2}\sum_{e=1}^{E} \{\mathrm{q}\}^T [k]\{q\} - \sum_{e=1}^{E} \{\mathrm{q}\}^T \{f^e\} - \sum_{e=1}^{E} \{\mathrm{q}\}^T \{T^e\} - \{Q\}^T \{F_c\} \\ &= \frac{1}{2}\{Q\}^T [K]\{Q\} - \{Q\}^T \{F\} \end{aligned} \quad \textbf{(5.34)}$$

where $[K\ ]$and $\{F\ \}$are

$$[K] = \sum_{e=1}^{E} [k] \qquad \{F\} = \sum_{e=1}^{E} \{f^e\} + \{T^e\} + \{F_c\} \quad \textbf{(5.35)}$$

Using the principle of minimum potential energy, the static equilibrium equations for the structure can be obtained.

$$\frac{\partial \Pi}{\partial Q_i} = 0 \qquad i = 1,2,3, \ldots N \quad \textbf{(5.36)}$$

The set of linear algebraic equations to solve nodal displacements are obtained by substituting equation **(5.34)** into equation **(5.36)** and shown below

$$[K]\{Q\} = \{F\} \quad \textbf{(5.37)}$$

The element stresses are calculated after solving the equation given above.

5.2.1.3. Stress calculations

Using the Hooke's law given in **(5.3)** and substituting equation **(5.15)** for the strain, the element stresses can be calculated by

$$\{\sigma\} = [E][B]\{q\} \quad \textbf{(5.38)}$$

Making use of equation **(5.38),** $\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{yz},$ and τ_{zx} stress components can be obtained, and by using them the three principal stresses are calculated as follows

$$\sigma_1 = \frac{I_1}{3} + c\cos\theta$$

$$\sigma_2 = \frac{I_1}{3} + c\cos(\theta + \frac{2\pi}{3}) \qquad \textbf{(5.39)}$$

$$\sigma_3 = \frac{I_1}{3} + c\cos(\theta + \frac{4\pi}{3})$$

where c and θ are given by

$$a = \frac{I_1^2}{3} - I_2$$

$$b = -2\left(\frac{I_1}{3}\right)^3 + \frac{I_1 I_2}{3} - I_3$$

$$\textbf{(5.40)}$$

$$c = 2\sqrt{\frac{a}{3}}$$

$$\theta = \frac{1}{3}\cos^{-1}\left(-\frac{3b}{ac}\right)$$

and the three invariants I_1, I_2, $I_3$of the stress tensor are

$$I_1 = \sigma_x + \sigma_y + \sigma_z$$

$$I_2 = \sigma_x\sigma_y + \sigma_y\sigma_z + \sigma_x\sigma_z - \tau_{xy}^2 - \tau_{yz}^2 - \tau_{xz}^2 \qquad \textbf{(5.41)}$$

$$I_3 = \sigma_x\sigma_y\sigma_z + 2\tau_{xy}\tau_{yz}\tau_{xz} - \sigma_x\tau_{yz}^2 - \sigma_y\tau_{xz}^2 - \sigma_z\tau_{xy}^2$$

5.2.1.4. Dynamic consideration

The Lagrangean is defined by

$$L = T - \Pi \qquad \textbf{(5.42)}$$

According to Hamilton's principle, in an arbitrary $t_1 - t_2$ time interval, the state of motion of a body extremizes the functional

$$I = \int_{t_1}^{t_2} L dt \qquad \textbf{(5.43)}$$

If generalized variables $(q_1, q_2, \dots, q_n, \dot{q}_1, \dot{q}_2, \dots, \dot{q}_n)$, where $\dot{q}_i = d\dot{q}_i/dt$, are used to express L, then the equations of motion are given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \quad i = 1\ to\ n \qquad \textbf{(5.44)}$$

The kinetic energy is given by

$$T = \frac{1}{2}\int_v \{\dot{u}\}^T\{\dot{u}\}\rho dV \qquad \textbf{(5.45)}$$

where density of the material is shown by ρ and velocity vector of the point at x with components $\dot{u}, \dot{v}, and\ \dot{w}$ is

$$\{\dot{u}\} = [\dot{u}, \dot{v}, \dot{w}]^T \qquad \textbf{(5.46)}$$

u can be expressed by using shape functions as follows

$$\{u\} = [N]\{q\} \qquad \textbf{(5.47)}$$

similarly the velocity vector is given by

$$\{\dot{u}\} = [N]\{\dot{q}\} \qquad \textbf{(5.48)}$$

using equations **(5.48)** and **(5.45)** the kinetic energy T_e in element e is

$$T_e = \frac{1}{2}\{\dot{q}\}^T\left[\int_e \rho[N]^T[N]dV\right]\{\dot{q}\} \qquad \textbf{(5.49)}$$

The bracketed expression in equation **(5.49)** is the element mass matrix

$$[m^e] = \int_e \rho[N]^T[N]dV \qquad \textbf{(5.50)}$$

This mass matrix is consistent with the shape functions chosen and is called the consistent mass matrix. On summing over all the elements

$$T = \sum_{e} T_e = \sum_{e} \frac{1}{2}\{\dot{q}\}^T[m^e]\{\dot{q}\} = \frac{1}{2}\{\dot{Q}\}^T[M]\{\dot{Q}\} \quad \textbf{(5.51)}$$

The potential energy is given by

$$\Pi = \frac{1}{2}\{Q\}^T[K]\{Q\} - \{Q\}^T\{F\} \quad \textbf{(5.52)}$$

Using the Lagrangean given in equation **(5.42)**, the equation of motion is obtained as follows

$$[\mathrm{M}]\{\ddot{Q}\} + [K]\{Q\} = \{F\} \quad \textbf{(5.53)}$$

The force F is zero for free vibrations. Thus,

$$[\mathrm{M}]\{\ddot{Q}\} + [K]\{Q\} = \{0\} \quad \textbf{(5.54)}$$

Considering the steady-state condition, starting from the equilibrium state, $\{Q\}$ can be taken as

$$\{Q\} = \{U\}\sin\omega t \quad \textbf{(5.55)}$$

where ω is the circular frequency and $\{U\}$ is the vector of nodal amplitudes of vibration. Substituting equation **(5.55)** into **(5.54)**

$$[K]\{U\} = \omega^2[M]\{U\} \quad \textbf{(5.56)}$$

This is the generalized eigenvalue problem

$$[K]\{U\} = \lambda[M]\{U\} \quad \textbf{(5.57)}$$

where $\{U\}$ is the eigenvector, which represents vibration mode for corresponding eigenvalue, and λ, the square of the circular frequency ω, is the eigenvalue. The frequency f in hertz (cycles per second) is obtained from

$$f = \frac{\omega}{2\pi} \quad \textbf{(5.58)}$$

Element mass matrix

The consistent mass matrix for tetrahedral element is obtained by using equation **(5.50)** and is given by

$$[m^e] = \frac{\rho V_e}{20}\begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ & & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ & & & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ & & & & 2 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ & & & & & 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ & & & & & & 2 & 0 & 0 & 1 & 0 & 0 \\ & & & & & & & 2 & 0 & 0 & 1 & 0 \\ & & & Symmetric & & & & & 2 & 0 & 0 & 1 \\ & & & & & & & & & 2 & 0 & 0 \\ & & & & & & & & & & 2 & 0 \\ & & & & & & & & & & & 2 \end{bmatrix} \qquad \textbf{(5.59)}$$

5.3. Object-Oriented Finite Element Analysis

The fundamental concepts in object-oriented programming is introduced in section 5.1 and an overview of finite element method with element formulations is given in section 5.2. In this section, the development of a new finite element solver by combining these two methodologies is presented.

5.3.1. EafeLib: A C++ finite element analysis library and its base classes

EafeLib is a collection of C++ classes designed to do finite element analysis in 3D with an object-oriented approach. It is built around six main classes: Node, Element, Load, BoundaryCondition, Material, and Model. It also has some auxiliary classes such as InputReader, Solver, and OutputWriter. The primary class in EafeLib solver is the Model class. Using UML class diagrams the relationship between these classes are shown in Figure 5.7.

In order to successfully use EafeLib solver for a finite element problem, an input file, which is consistent with the EafeLib's predefined input file format, must be provided. EafeLib's input file format contains the nodal coordinates, element connectivities, boundary conditions, loads and material definitions. An InputReader class object reads this input file and creates corresponding Node, BoundaryCondition, Load and Material objects. Subsequently, Model class uses these objects along with a vector, obtained from InputReader and filled with element connectivities, to create elements, calculate element stiffness matrices and assemble them into the global stiffness matrix. In addition, the force terms for each element is calculated and assembled into the global force vector. The Global stiffness matrix

can be stored in two different matrix types: dense matrix type and sparse matrix type. In dense matrix all of the elements of the stiffness matrix is stored, whereas in sparse matrix only the nonzero elements are stored. The matrices that arise from the discretization of a three-dimensional domain are inherently sparse matrices and storing them in a dense matrix format requires a considerable amount of memory. Therefore, the global stiffness matrix in EafeLib solver is stored in a sparse matrix format by default, and the inclusion of dense matrix format is just for illustration purposes.

A Solver class object obtains the global stiffness matrix and global force vector from a Model class object, than solves the linear system and returns the displacements. Finally, an OutputWriter class object takes the displacement results and writes them into a text file. The procedure described so far is a berief overview of the EafeLib solver structure and the detailed base class explanations are given in the following sections.

5.3.1.1. Element class

Element class in EafeLib solver is created as an abstract class and includes declarations of pure functions that must be implemented in the derived classes. The UML class diagram for the element class is given in Figure 5.8. Most of the methods shown in the figure does not have an implementation in the element class. A derived class, for example Tet4 , must have implementations for these pure functions.

5.3.1.2. Node class

A Node class object mainly holds x, y, z coordinates, loads and boundary conditions defined on the node, displacements, and nodal stresses as shown in the Figure 5.9.

A class such as Tet4 that is derived from Element class will have a number of Node class objects as its data members. In Tet4 class, for example, there are four node objects associated with the four nodes of the tetrahedral element.

Making use of Node class's member functions, global node number of an element's local node, along with the x, y, and z coordinates with corresponding displacement values can be acquired inside the element class. Moreover, loads and boundary conditions defined on a node object that is a member of the aforementioned element class are also become accessible.

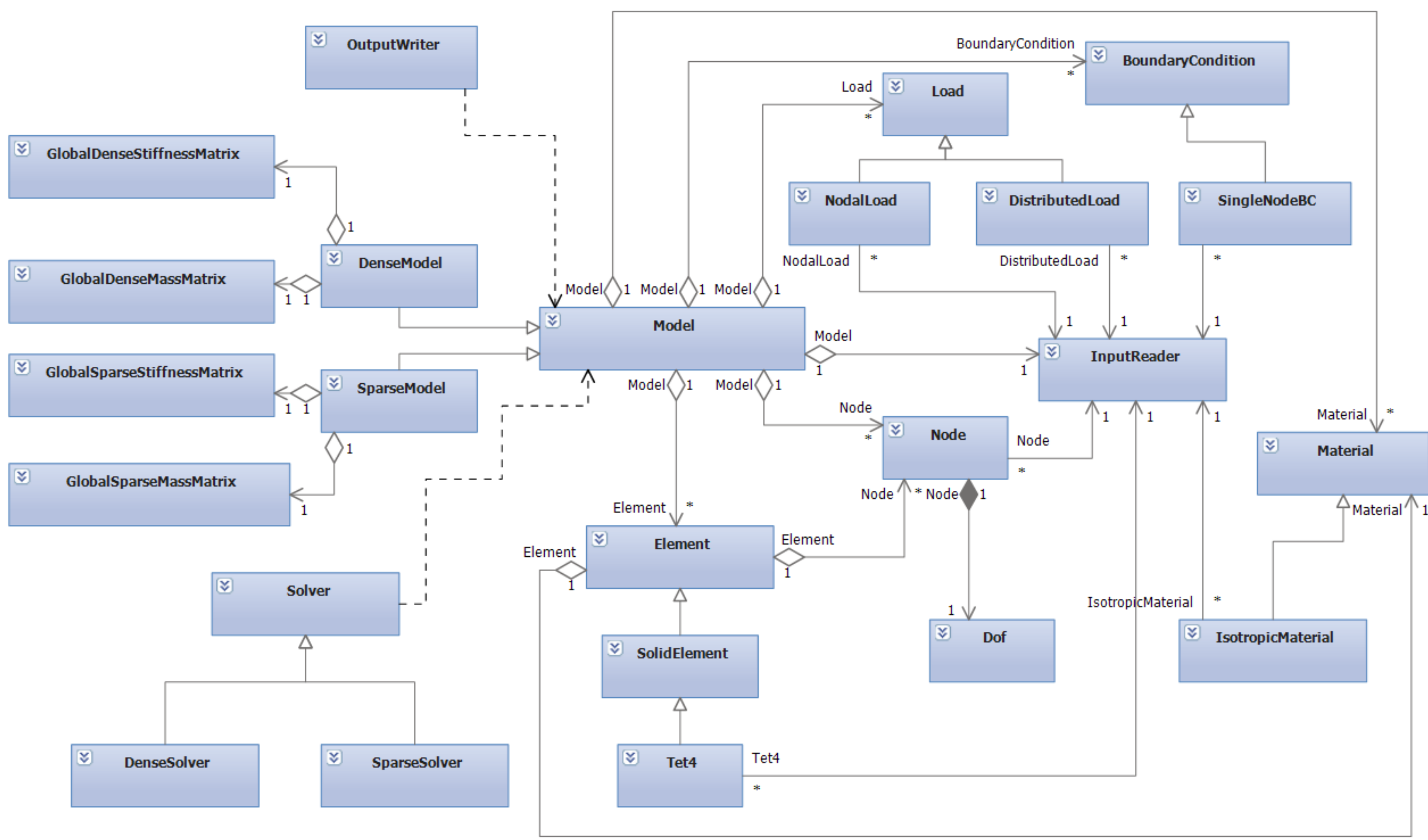


Figure 5.7 : The simplified UML class diagrams of EafeLib solver.

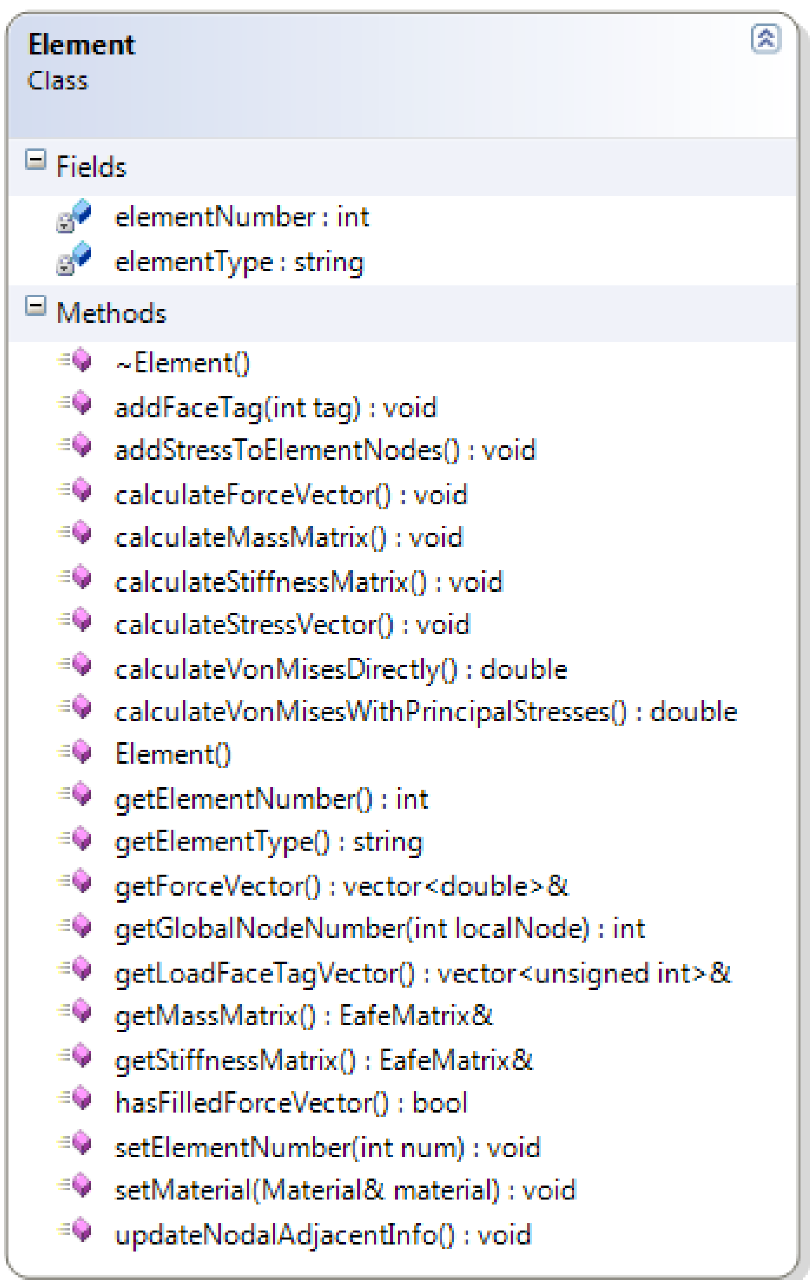


Figure 5.8 : The UML representation of the element class.

5.3.1.3. Material class

Material class is an abstract base class and IsotropicMaterial class is a more specific class derived from material class as shown in Figure 5.10. IsotropicMaterial class's data members are density, poisson ration, shear modulus, and young modulus. In addition, IsotropicMaterial class has set and get functions to manipulate its data members. The material data obtained from the input file are stored in the objects of the IsotropicMaterial class. Element class is designed to have objects of classes that are derived from Material class as its data members. Hence, it is possible to store different material data and even different material type for each element.

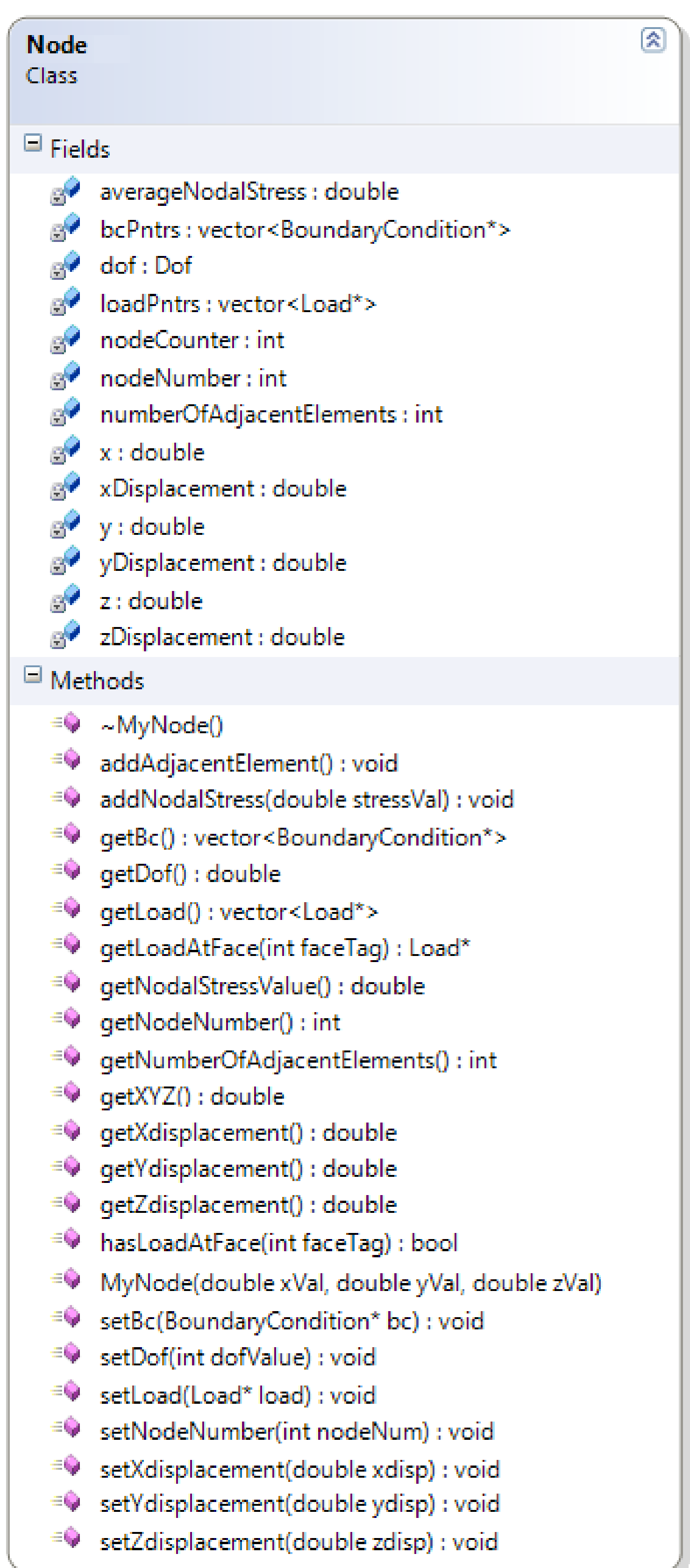


Figure 5.9 : The UML representation of the node class.

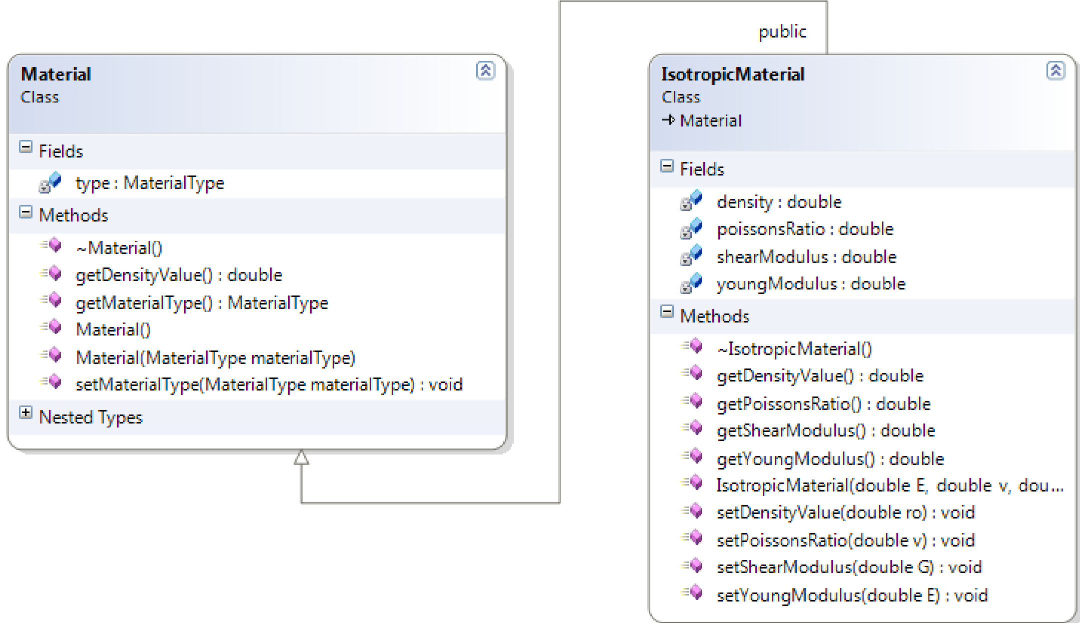


Figure 5.10 : The UML representation of Material and IsotropicMaterial classes.

5.3.1.4. Load class

The loads applied to the system are held in the objects of the subclasses of Load class such as NodalLoad class and DistributedLoad class as shown in Figure 5.11. Both Node and Element classes have pointers to the load objects, due to the fact that, a load can be applied either on a single node or on an element. NodalLoad class deals with loads applied on a single node, whereas DistributedLoad class deals with loads such as pressure or surface traction. Another load class, such as body load, can easily be created by subclassing load class.

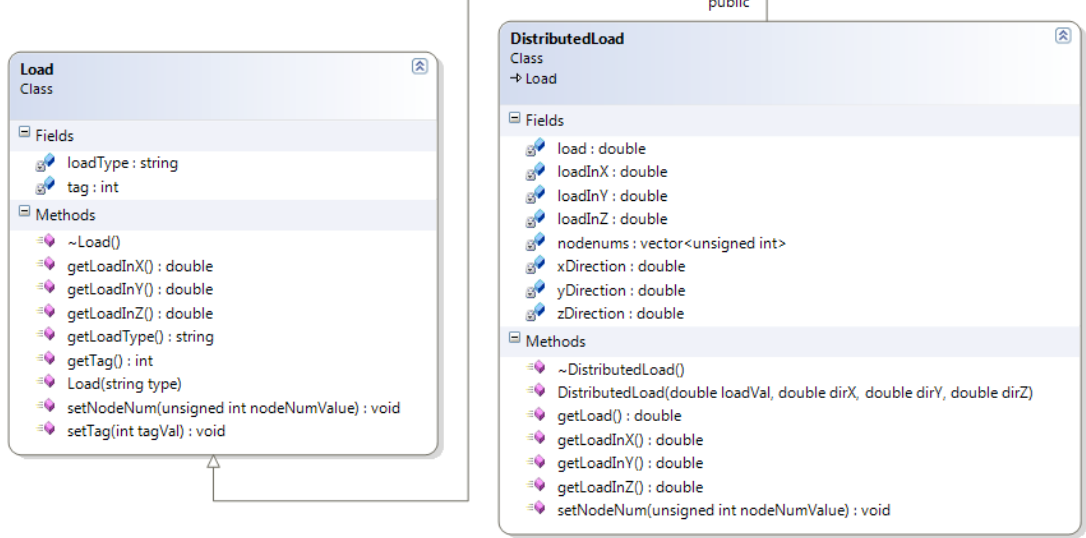


Figure 5.11 : The UML representation of the Load and the DistributedLoad classes.

5.3.1.5. Boundary condition class

BoundaryCondition class is designed as an abstract base class and it is used to create subclasses such as SingleNodeBC. The UML class diagrams with data members and member functions for these two classes are given in Figure 5.12. The figure shows that by using SingleNodeBC class's member functions such as setU, setV, and SetW, it is possible to set u,v and w displacement components of a boundary condition object. Subsequently, boundary condition for any node can be defined by incorporating a pointer to this boundary condition object into the node object .

5.3.2. Global stiffness and mass matrices assembly process

The two fundamental components of the finite element model are the global stiffness and global mass matrices.The assemblage of the global stiffnes and mass matrices are accomplished by Model class's member functions as it is explained in the following section.

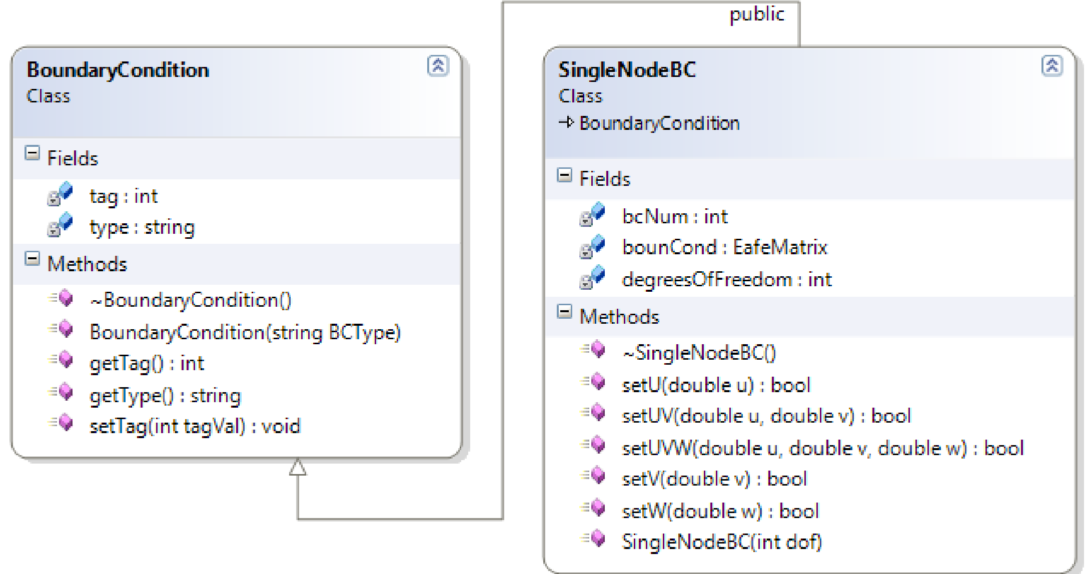


Figure 5.12 : The UML representation of the Boundary Condition classes.

5.3.2.1. Model class

The Model class is the most important class in the EafeLib FEM solver library and it holds the global stiffness and global mass matrices of the problem. Figure 5.13 shows the UML class diagram with data members and member functions of the SparseModel class which is a subclass of the base Model class. The SparseModel class has everything that is necessary to describe a finite element problem such as

global stiffness matrix, global load vector, and an empty global displacements vector which is filled by Solver class after solving linear system of equations.

The function assembleGlobalStiffnessAndMassMatrices (bool assembleMassMatrix) of SparseModel class does exactly what its name suggests and it is explained below.

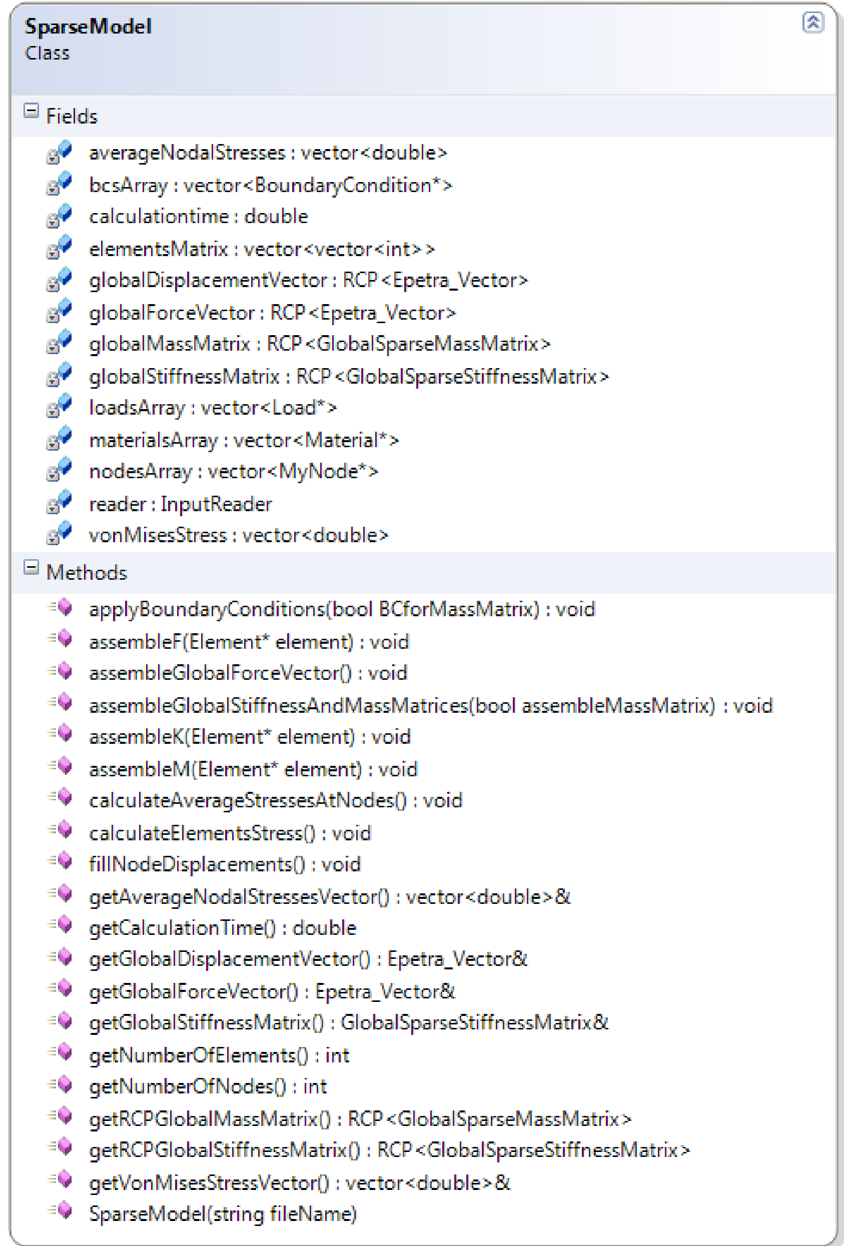


Figure 5.13 : The UML representation of the SparseModel class.

The UML activity diagram for assembleGlobalStiffnessAndMassMatrices function is given in Figure 5.14. Calculating mass matrices for each element and assembling them into a redundant global mass matrix is a waste of computer processor and memory. To avoid this, the function begins with an if statement to check whether the

mass matrix is required for the problem, and the mass matrix is initialized only if it is necessary.

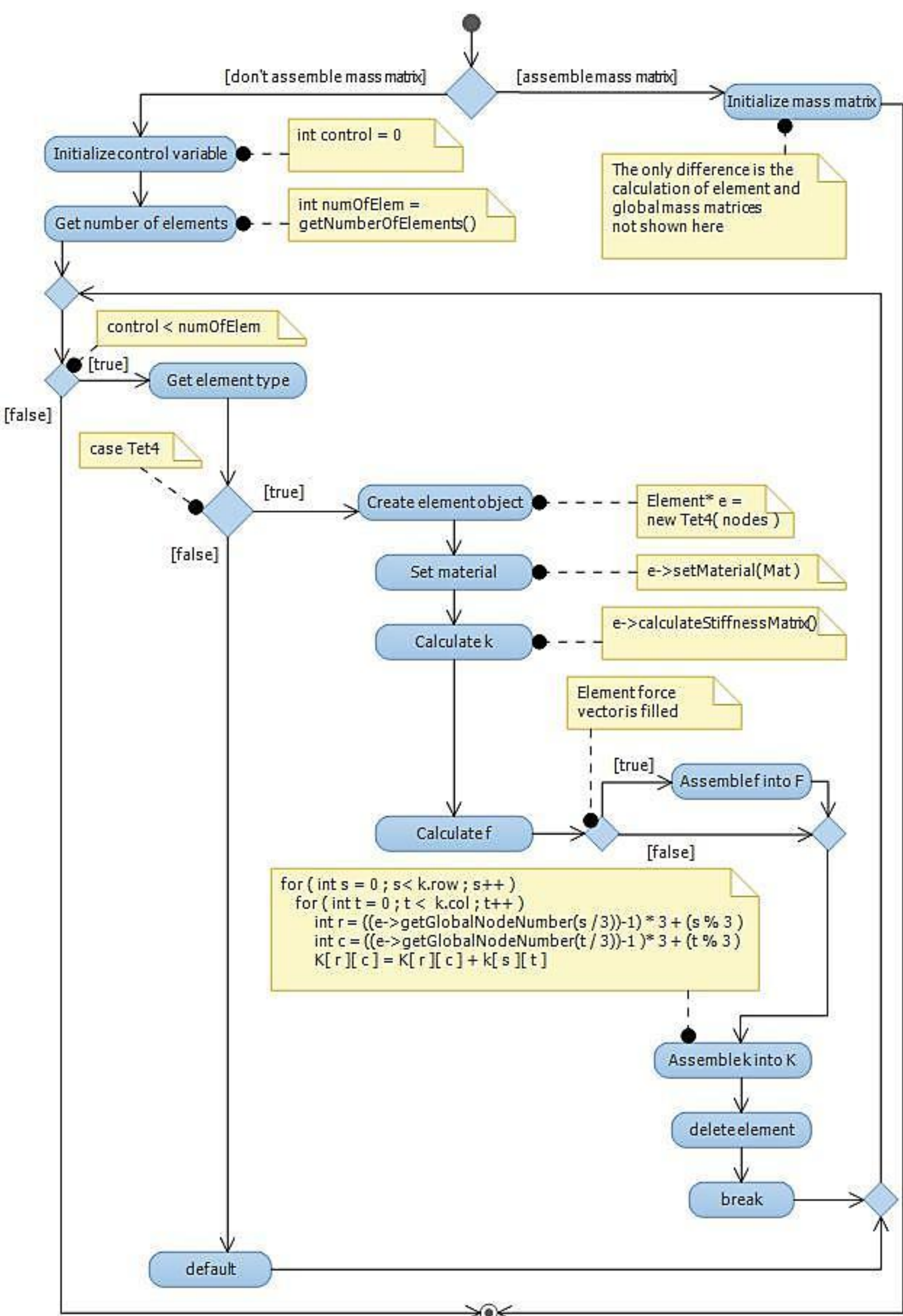


Figure 5.14 : The UML activity diagram of the assembler function.

After the mass matrix check, the assembler function enters into an iteration over the element data matrix, which is a matrix created by InputReader class and includes the required data (e.g. element type, nodes, and material) to create element objects. The

function gets an integer value that represents the element type from the second column of each row of the matrix and by using a switch statement it creates a corresponding element object. Subsequently, for the created element the material is set, stiffness matrix and element force vector are calculated. Afterward, element stiffness matrix is assembled into the global stiffness matrix and if the element has a filled force vector, this vector is assembled into the global force vector. Before leaving the switch statement the created element is deleted to free the allocated memory. The same procedure is repeated for each element until the global stifness and mass matrices are completely assembled.

5.3.3. Linear algebra library

Most of the calculations in the finite element method such as solving linear system of algebraic equations given in **(5.60)** or generalized eigenvalue problem given in **(5.61)** are done by using matrices.

$$\{F\} = [K]\{d\} \quad \textbf{(5.60)}$$

$$[K]\{A\} = \lambda[M]\{A\} \quad \textbf{(5.61)}$$

Unfortunately, C++ language does not have an in-built matrix library. Users are encouraged either to develop their own matrix classes or rely on third party linear algebra packages. A list of well-known open source linear algebra packages are given in Table 5.1. Two of the libraries given in the table are more professionally developed and widely used: Trilinos and PETSc. In comparison to these two, the other libraries given in the table are rather small libraries.

5.3.3.1. PETSc

PETSc, the portable, extensible toolkit for scientific computation, is a suite of data structures and routines in C language for the parallel solution of scientific applications modeled by partial differential equations. It supports Message Passing Interface (MPI), and is developed by Argonne National Laboratory of University of Chicago [44].

Table 5.1 : Open source linear algebra packages.

Package	Language
Trilinos	C++
PETSc	C
Eigen	C++
Armadillo	C++
MTL	C++
Blitz++	C++

5.3.3.2. Trilinos

The Trilinos project is a collection of open source linear algebra packages developed by a team at the Sandia National Laboratories for the solution of large-scale, complex multi-physics engineering and scientific problems [45]. Trilinos package's capabilities include constructing and using vectors, dense and sparse matrices, iterative and direct solution of linear systems and solution of nonlinear, eigenvalue and time dependent problems. Moreover, unlike PETSc, Trilinos is developed in C++ language with an object oriented software framework. Therefore, Trilinos is selected as linear algebra package in EafeLib solver in which it is utilized to create large matrices and to solve linear system of equations as well as eigenvalue problems.

5.3.4. Input and output file formats

EafeLib solver is designed to read an input file, to process it and to write the results in an output file. EafeLib solver's input and output file formats is given below.

5.3.4.1. Input file

An example input file is given in Figure 5.15. It contains nodes, elements, materials, boundary conditions, and loads sections. In each section the first line defines the total number of entries in the section and the last line indicates the end of the section.

In the nodes section, each node is described by a line which contains x, y, z coordinates, boundary conditions, and loads data for the node. If there is no defined load or boundary condition for a node, it is shown by a zero value.

```
------------------------------------------------------------------------
NODES
------------------------------------------------------------------------
%NumberOfNodes
621
              x          y          z          bc        load
%Node
 1            0          0          0          29 0      0
 2            0          0          0.25       29 0      0
 ...          ...        ...        ...        ...       ...
 619          2.3083     3.0800     0.1163     0         0
 620          1.3955     1.9918     0.125      0         0
 621          7.7994     1.0015     0.1249     0         0
%EndNode
------------------------------------------------------------------------
ELEMENTS
------------------------------------------------------------------------
%NumberOfElements
2230
          Type     Material   Nodes                              load
%Element
 1        4        1          25      466     18      528        0
 2        4        1          395     123     111     464        2  30  0
 3        4        1          190     22      253     463        0
 ...      ...      ...        ...     ...     ...     ...        ...
 2228     4        1          325     297     457     576        0
 2229     4        1          234     251     220     445        0
 2230     4        1          188     171     157     445        0
%EndElement
------------------------------------------------------------------------
MATERIALS
------------------------------------------------------------------------
%NumberOfMaterials
1
              Type        E          v          rho
%Material
1             Isotropic   29000000   0.29       1
%EndMaterial
------------------------------------------------------------------------
BOUNDARY CONDITIONS
------------------------------------------------------------------------
%NumberOfBoundaryConditions
1
                    Type   u     v     w
%BoundaryCondition
29                  1      0     0     0
%EndBoundaryCondition
------------------------------------------------------------------------
LOADS
------------------------------------------------------------------------
%NumberOfLoads
1
        Type   x     y     z     Load
%Load
30      2      1     0     0     1000
%EndLoads
```

Figure 5.15 : EafeLib solver input file format.

Similarly, in the elements section each entry defines a new element by providing element type, material, nodes and load data defined for the element. Materials section

contains the defined materials with modulus of elasticity, poissons ratio and density values. Boundary conditions and loads sections contain boundary conditions and loads defined for the system respectively.

5.3.4.2. Output file

OutputWriter class in EafeLib solver provides several output files for displacement and stress calculation results. Figure 5.16 shows an example output file for u, v, and w nodal displacements, and Figure 5.17 shows an example output file for element von Mises stress values.

```
--------------------------------------------------------------------------------
DISPLACEMENTS
--------------------------------------------------------------------------------
%Nodes
                u               v               w
1               -1.40167e-012   -6.02496e-013   6.5739e-013
2               -1.81538e-012   -5.99436e-013   -7.54723e-013
3               1.34795e-012    -5.73226e-013   -6.16082e-013
4               1.7111e-012     6.15215e-013    -7.16354e-013
5               -0.0464359      -0.702057       7.14397e-005
6               -0.0464268      -0.702054       6.9327e-005
7               0.0465672       -0.702081       8.03487e-005
8               0.046576        -0.702078       6.4535e-005
...             ...             ...             ...
22305           0.000534384     -0.444953       5.11811e-005
22306           0.0431585       -0.667012       6.93339e-005
22307           0.0368602       -0.287516       -7.32378e-005
22308           0.00417458      -0.51658        5.70998e-005
22309           -0.00751449     -0.0101135      0.00134165
22310           0.020127        -0.372122       -5.9786e-005
22311           -0.0368561      -0.247867       -0.00018198
%EndNodeData
```

Figure 5.16 : EafeLib solver displacements output file format.

```
-----------------------------------------------------------------------------------------------------------------------
STRESS
-----------------------------------------------------------------------------------------------------------------------
%Elements                    von Mises
1                            50.2329
2                            11.1298
3                            84.5721
4                            0.705486
5                            21.652
6                            3.27621
7                            1.55988
8                            100.364
9                            78.9913
10                           34.5102
...                          ...
121903                       12.7636
121904                       13.3287
121905                       85.2865
121906                       84.1093
121907                       79.6254
121908                       7.87697
121909                       7.85883
121910                       7.07428
$EndElementData
```

Figure 5.17 : EafeLib solver stress output file format.

6. RESULTS AND DISCUSSION

6.1. Application Tests

In this section some example problems are solved by using EAFE software. The solutions are compared with the results obtained from a commercial FEM software: Abaqus.

6.1.1. A loaded cantilever beam

The problem is graphically represented in Figure 6.1. In order to find nodal displacements and element von Mises stress values for this problem, a model that contains 122833 linear tetrahedrons is used in EAFE software as given in Appendix A. Likewise, a model that contains 121205 linear tetrahedral elements is used in Abaqus software. EAFE's displacement and stress results are show in Figure 6.2 and Figure 6.4 respectively. Similarly Figure 6.3 and Figure 6.5 shows the Abaqus displacement and stress values. These figures clearly show that the results obtained from EAFE and Abaqus software are almost identical.

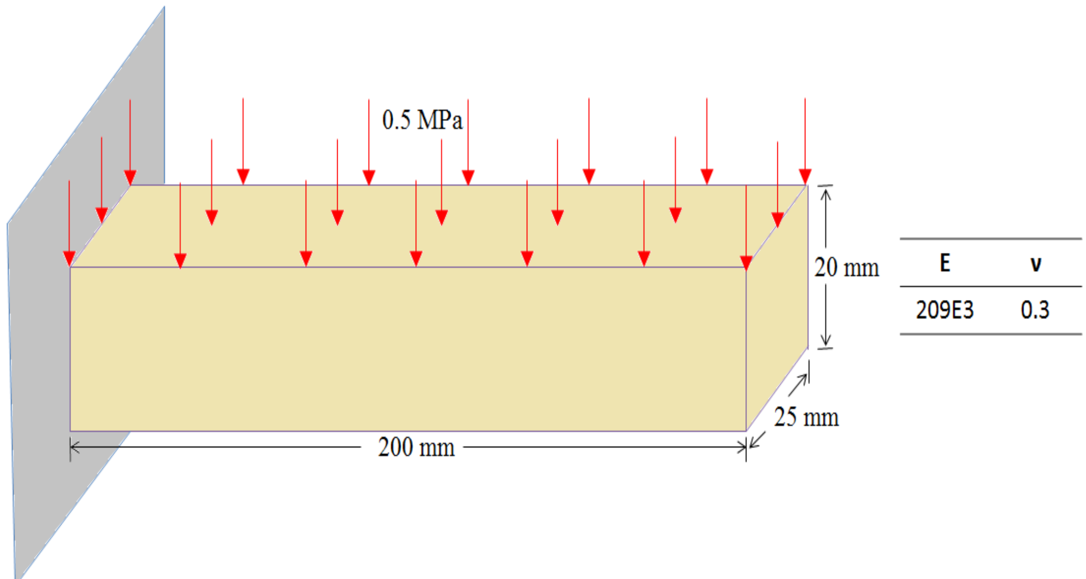


Figure 6.1 : A cantilever beam with a uniform load.

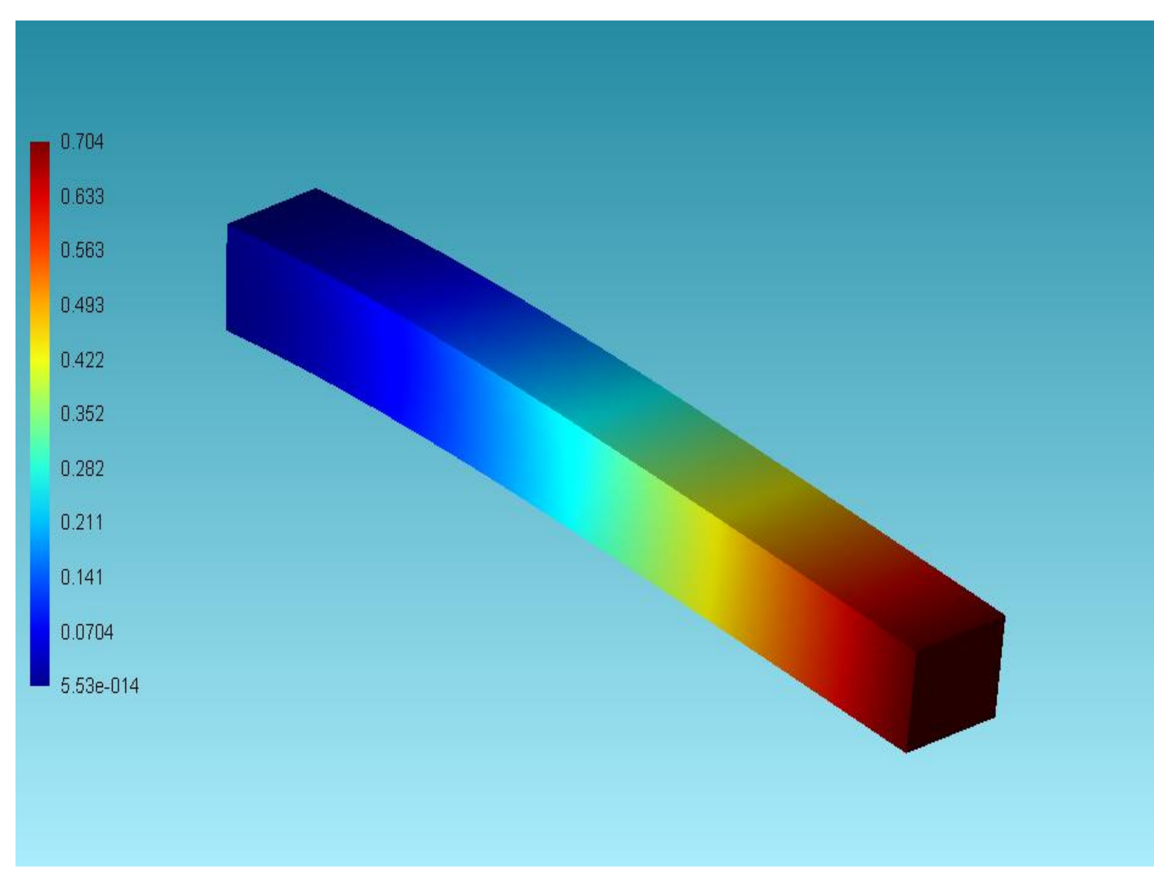


Figure 6.2 : Cantilever beam deformation contours in EAFE.

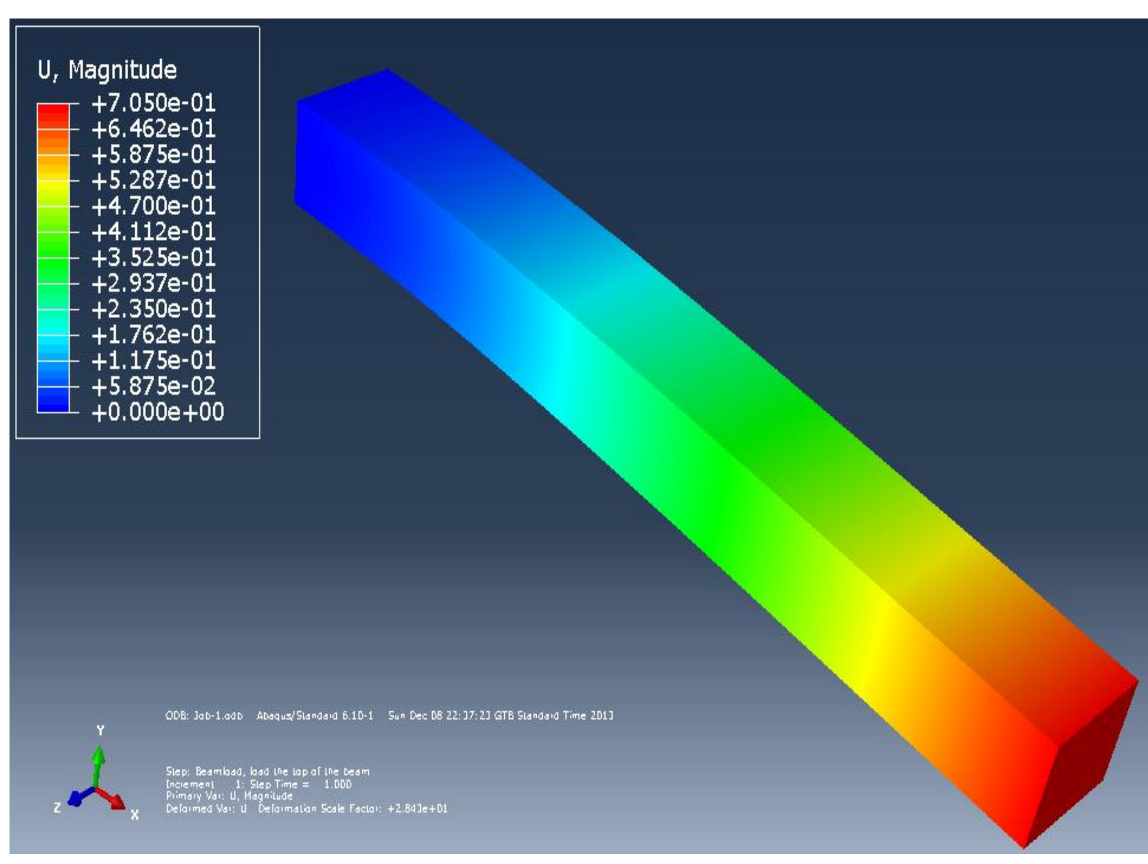


Figure 6.3 : Cantilever beam deformation contours in Abaqus.

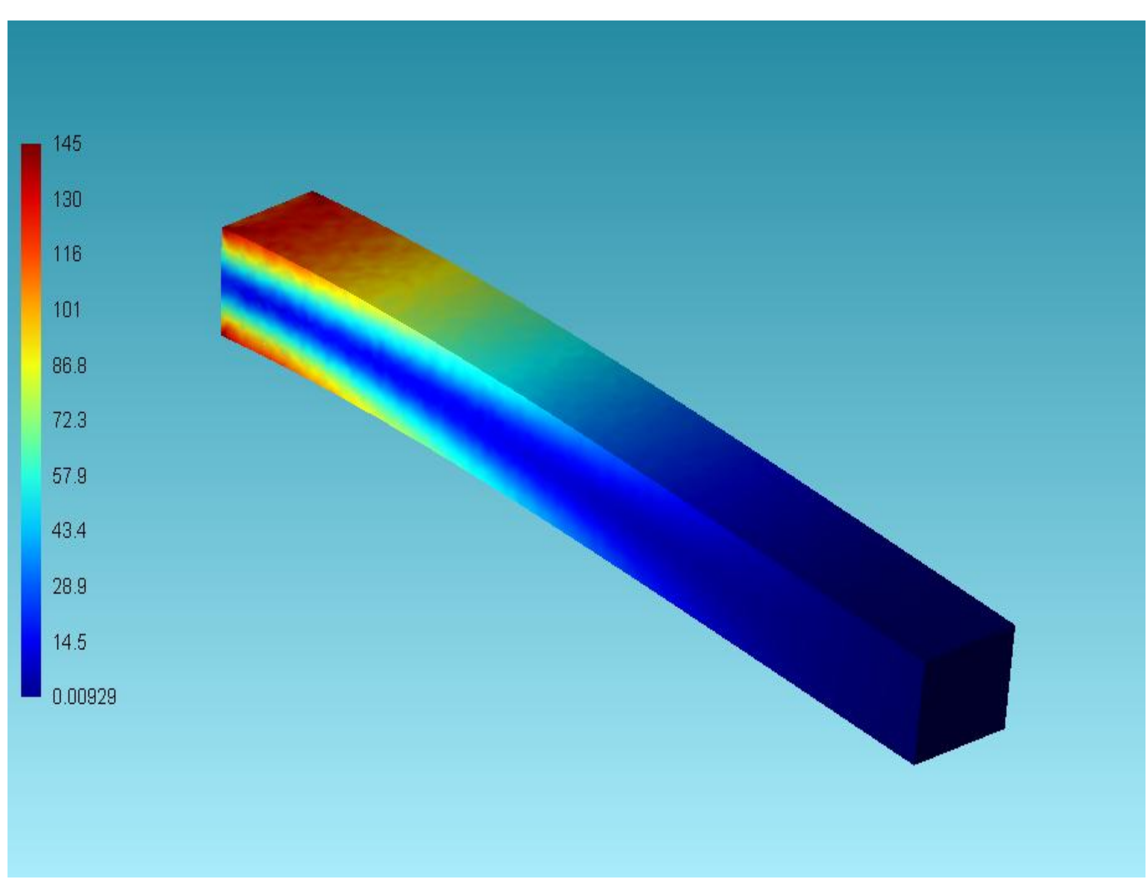


Figure 6.4 : Cantilever beam Mises stress contours in EAFE.

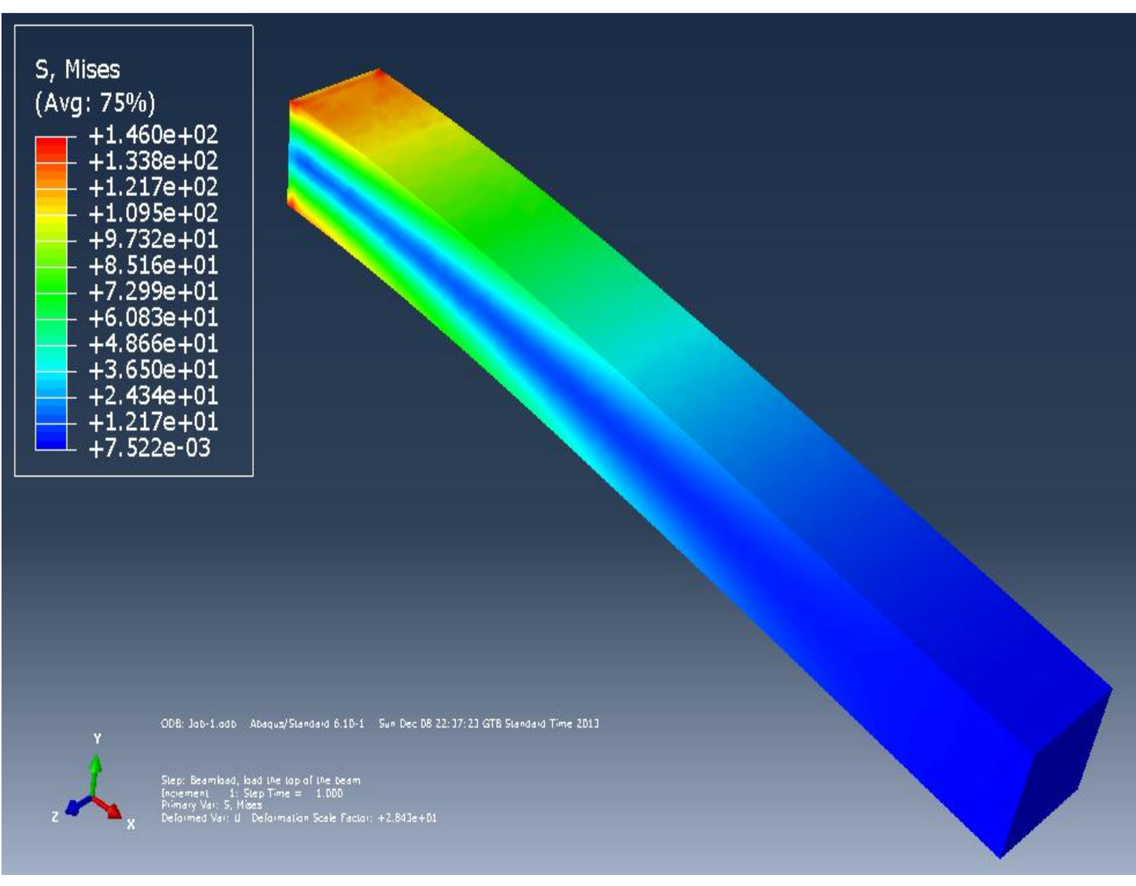


Figure 6.5 : Cantilever beam Mises stress contours in Abaqus.

Cantilever beam example is solved for different element numbers in both EAFE and Abaqus software. The maximum displacement and stress results from six analyses are summarized in Table 6.1.

Table 6.1 : Displacements and stress results with different element numbers.

Number of Elements		Max Displacement (mm)		Max von Mises (MPa)	
EAFE	Abaqus	EAFE	Abaqus	EAFE	Abaqus
93857	94725	0.701	0.702	139	142
54626	53960	0.694	0.695	132	134
37947	36080	0.687	0.686	128	130
14456	13601	0.664	0.658	120	120
4111	3834	0.604	0.593	106	116
2015	1919	0.554	0.560	97.6	108

6.1.2. A plate with a hole

The problem is graphically represented in Figure 6.6 and EAFE model for this problem is given in Appendix B. Different from the previous problem a mesh of 315735 tetrahedrons is used in both software. Nodal displacement contours obtained from EAFE and Abaqus are given in Figure 6.7 and Figure 6.8 respectively. Also, von Misses stress contours are given in Figure 6.9 for EAFE software, and in Figure 6.10 for Abaqus software. It is shown that same results are obtained from EAFE and Abaqus.

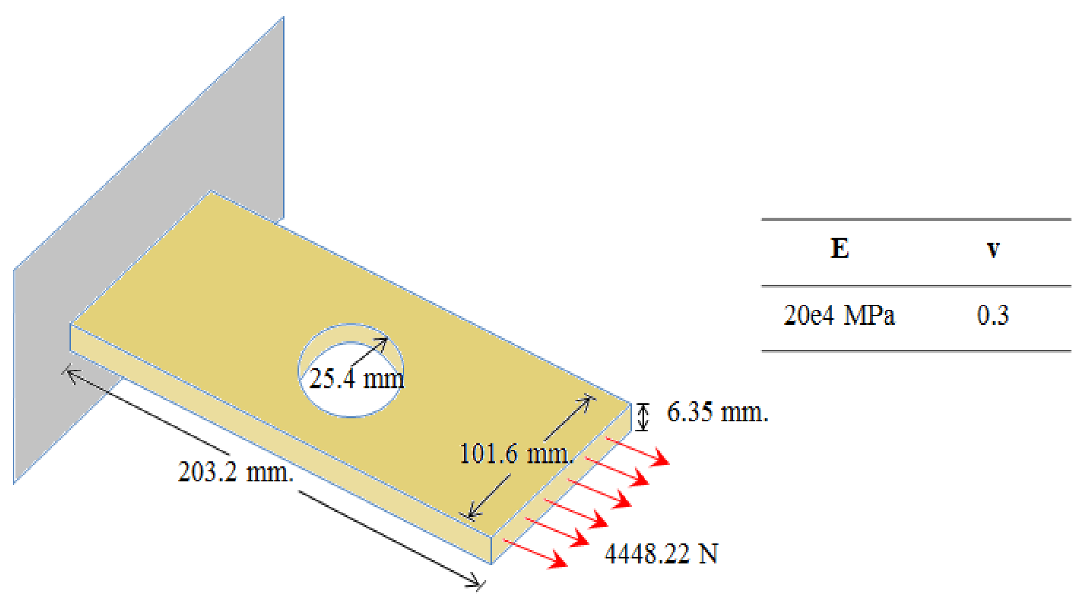


Figure 6.6 : A plate with a hole.

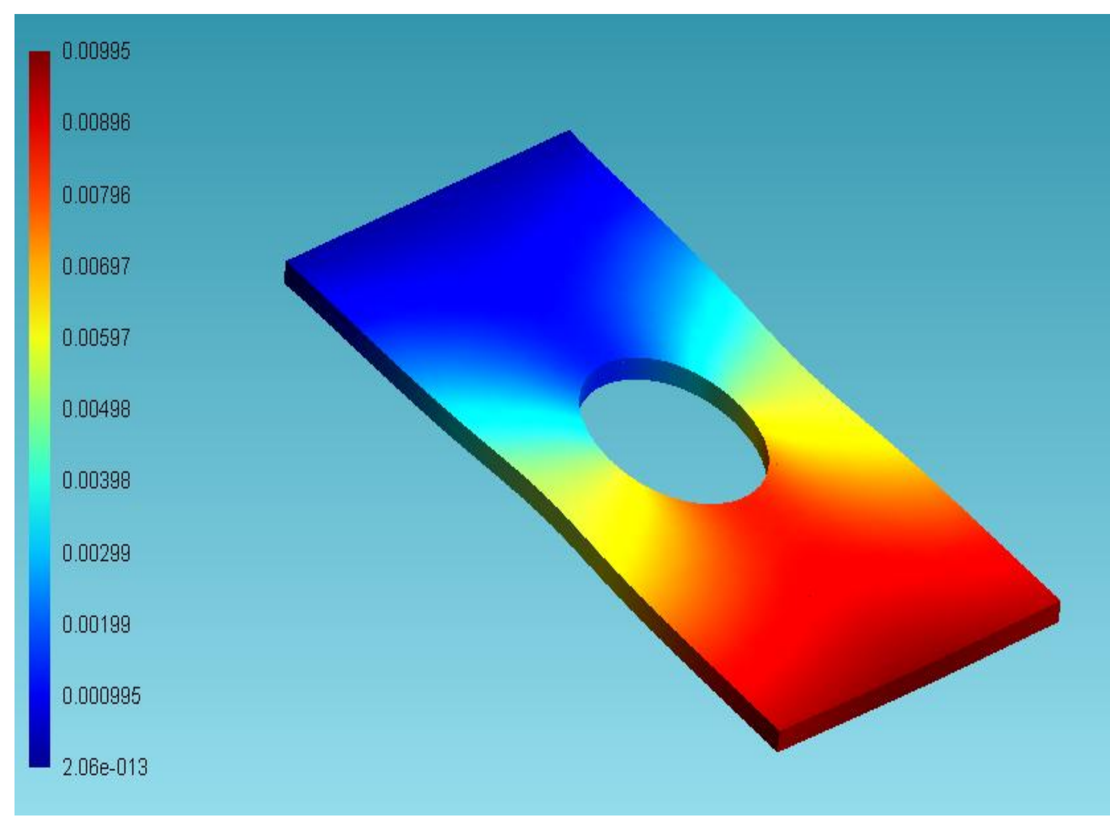


Figure 6.7 : Plate deformation contours in EAFE.

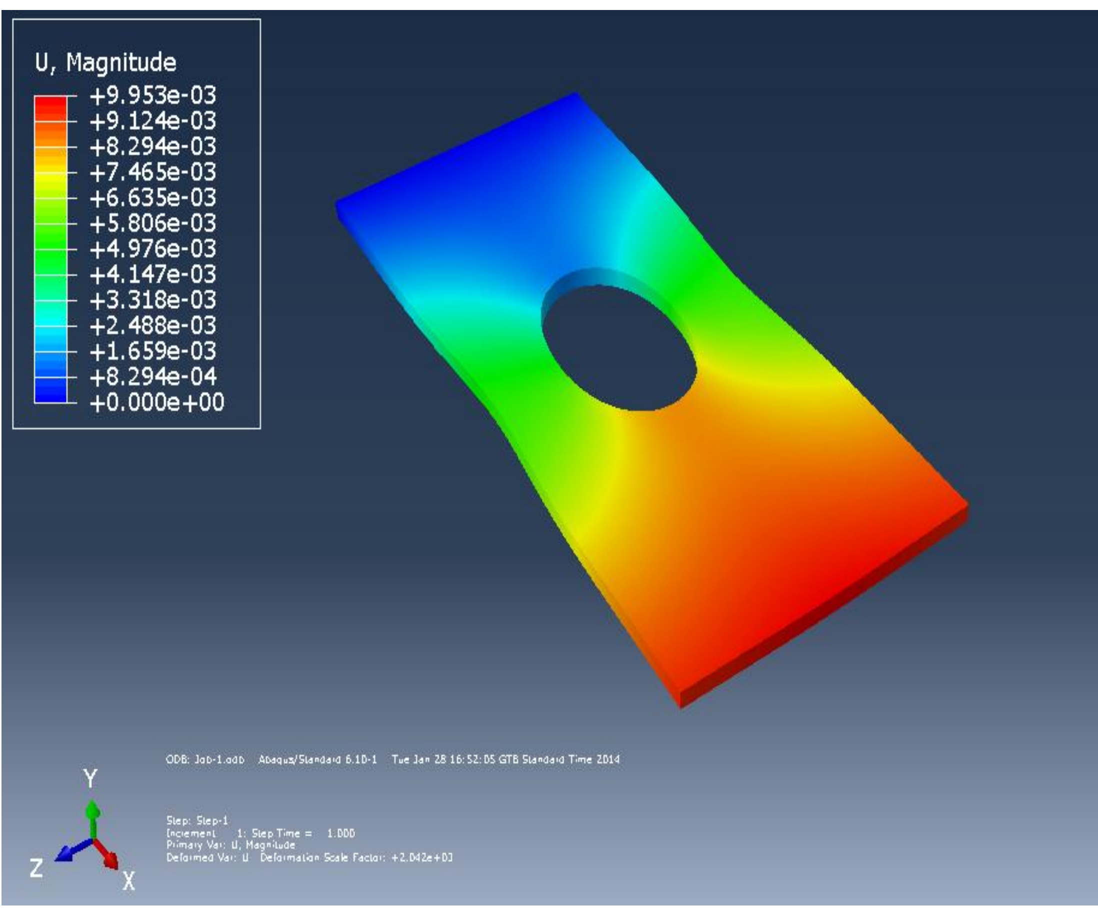


Figure 6.8 : Plate deformation contours in Abaqus.

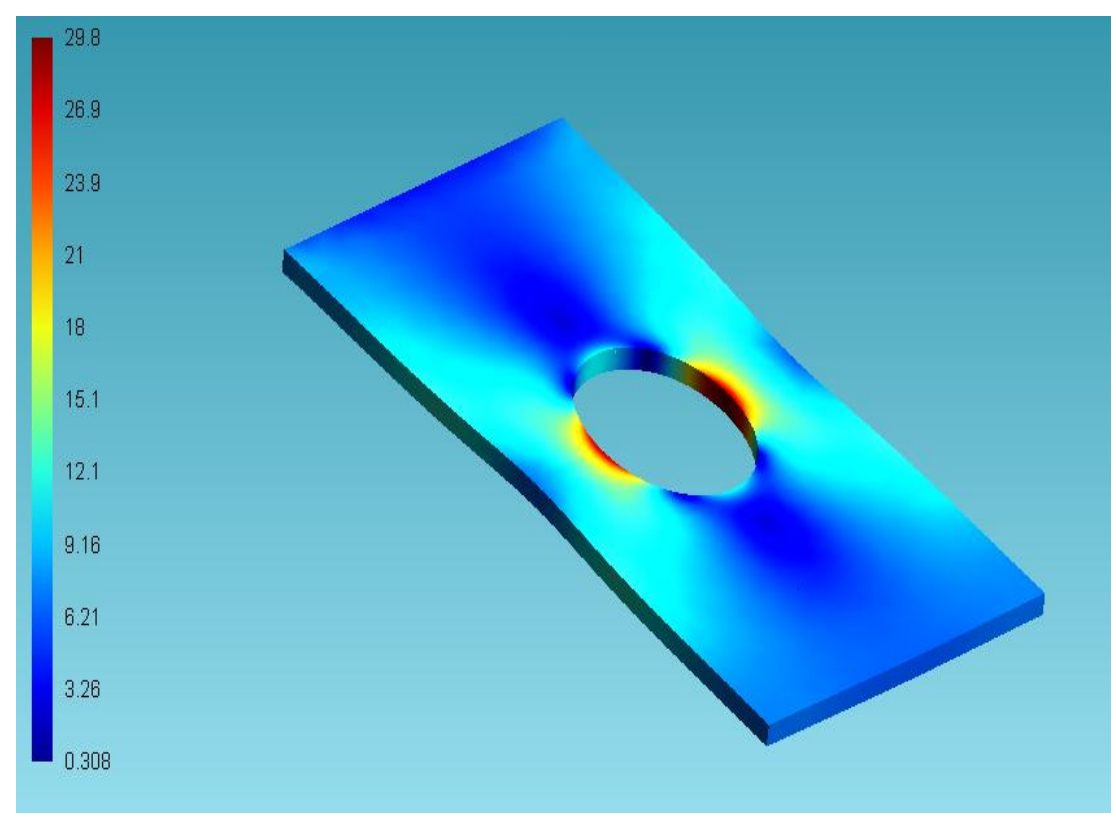


Figure 6.9 : Plate Mises contours in EAFE.

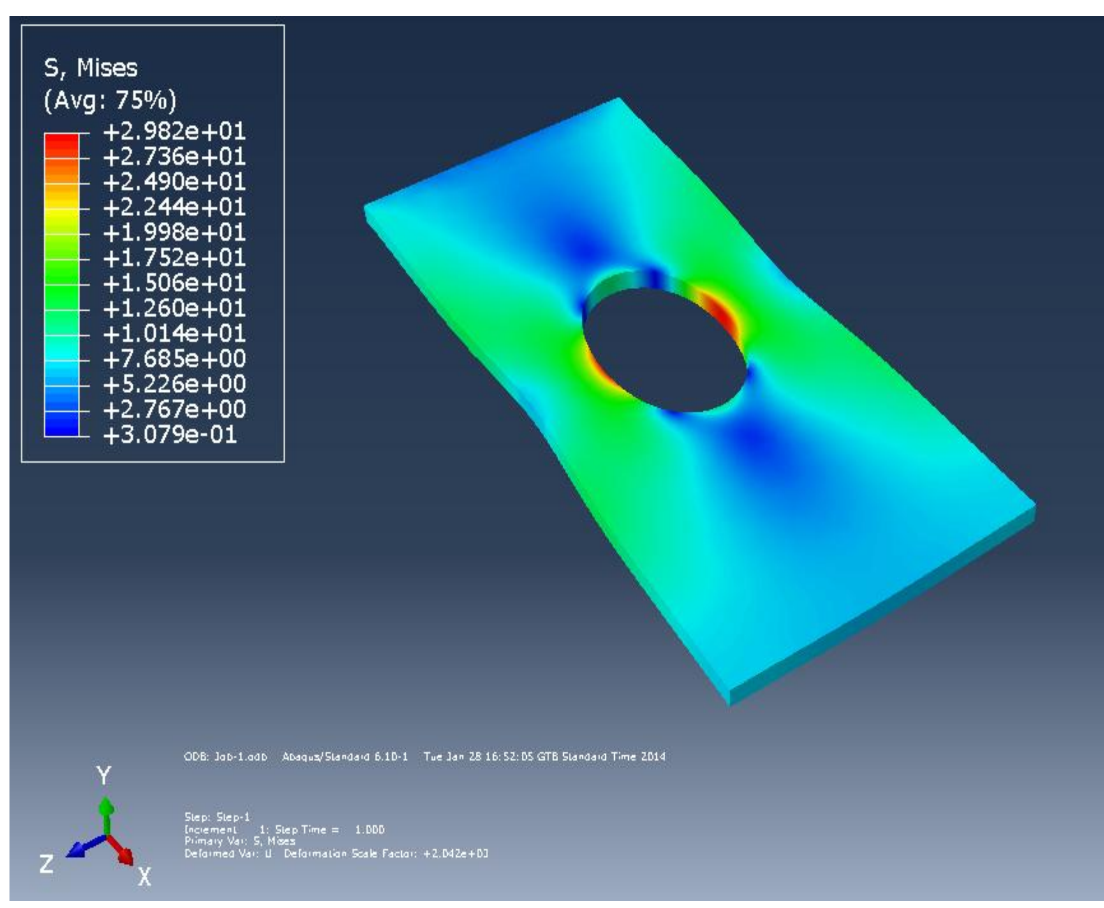


Figure 6.10 : Plate Mises contours in Abaqus.

The plate with a hole example is solved by using same mesh in both EAFE and Abaqus software. The maximum displacement and stress results from six analyses are summarized in Table 6.2.

Table 6.2 : Displacements and stress results with different element numbers.

Number of Elements		Max Displacement (mm)		Max von Mises (MPa)	
EAFE	Abaqus	EAFE	Abaqus	EAFE	Abaqus
201019	201019	0,009957	0,0099568	29,716	29,723
33803	33803	0,009931	0,0099314	29,371	29,357
26866	26866	0,009931	0,0099314	28,199	28,199
13981	13981	0,009881	0,0098806	25,855	25,834
7928	7928	0,009830	0,0098298	24,683	24,683
2729	2729	0,009804	0,0098044	23,304	23,304

6.1.3. A support beam with a uniform pressure

The problem is graphically represented in Figure 6.11 and EAFE model for this problem is given in Appendix C. Similar to the previous examples nodal displacement contours obtained from EAFE and Abaqus by using 133552 tetrahedral elements are given in Figure 6.12 and Figure 6.13 respectively. Also, von Misses stress contours are given in Figure 6.14 for EAFE software, and in Figure 6.15 for Abaqus software. It is shown that results from EAFE and Abaqus are similar.

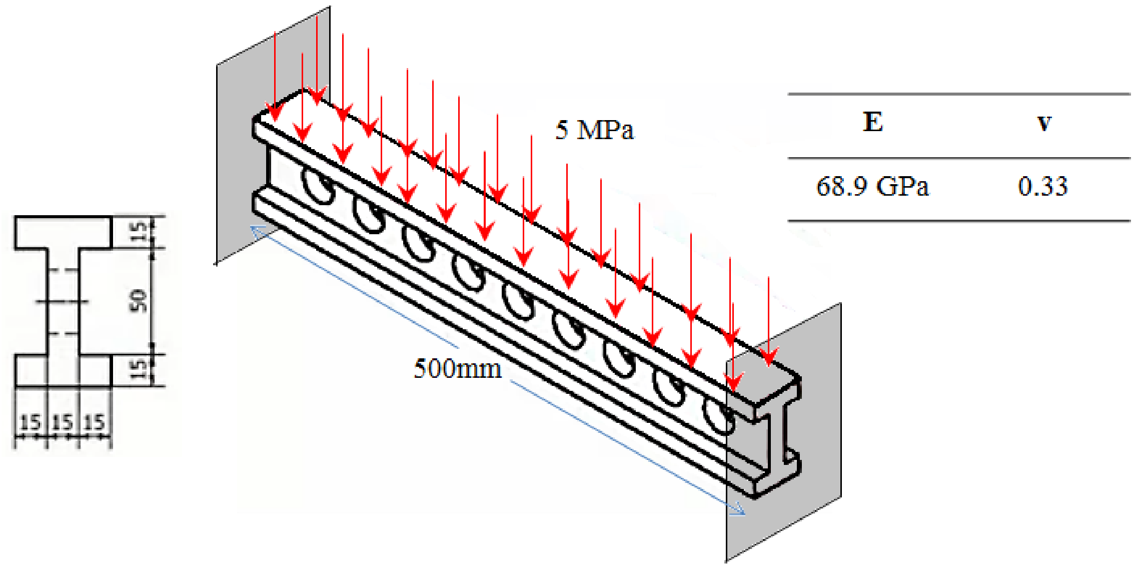


Figure 6.11 : Support beam.

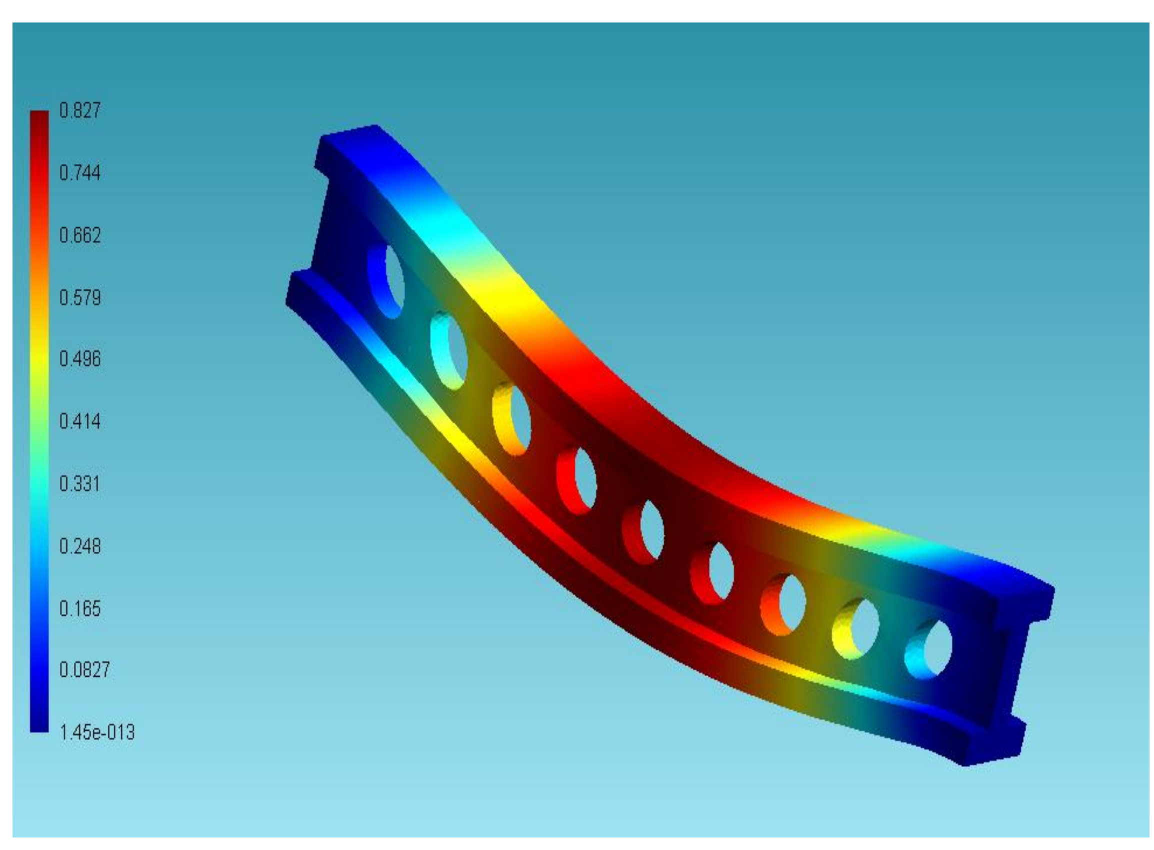


Figure 6.12 : Support beam displacement contours in EAFE.

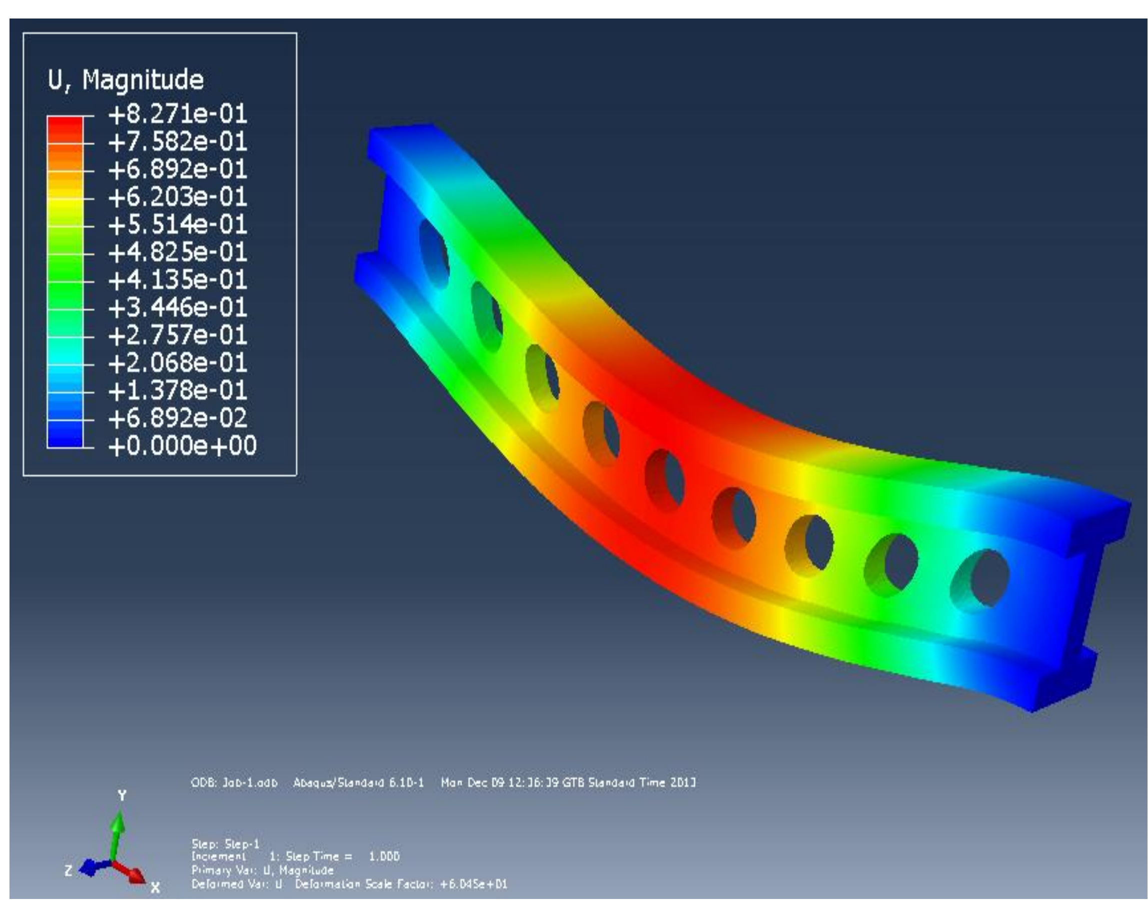


Figure 6.13 : Support beam displacement contours in Abaqus.

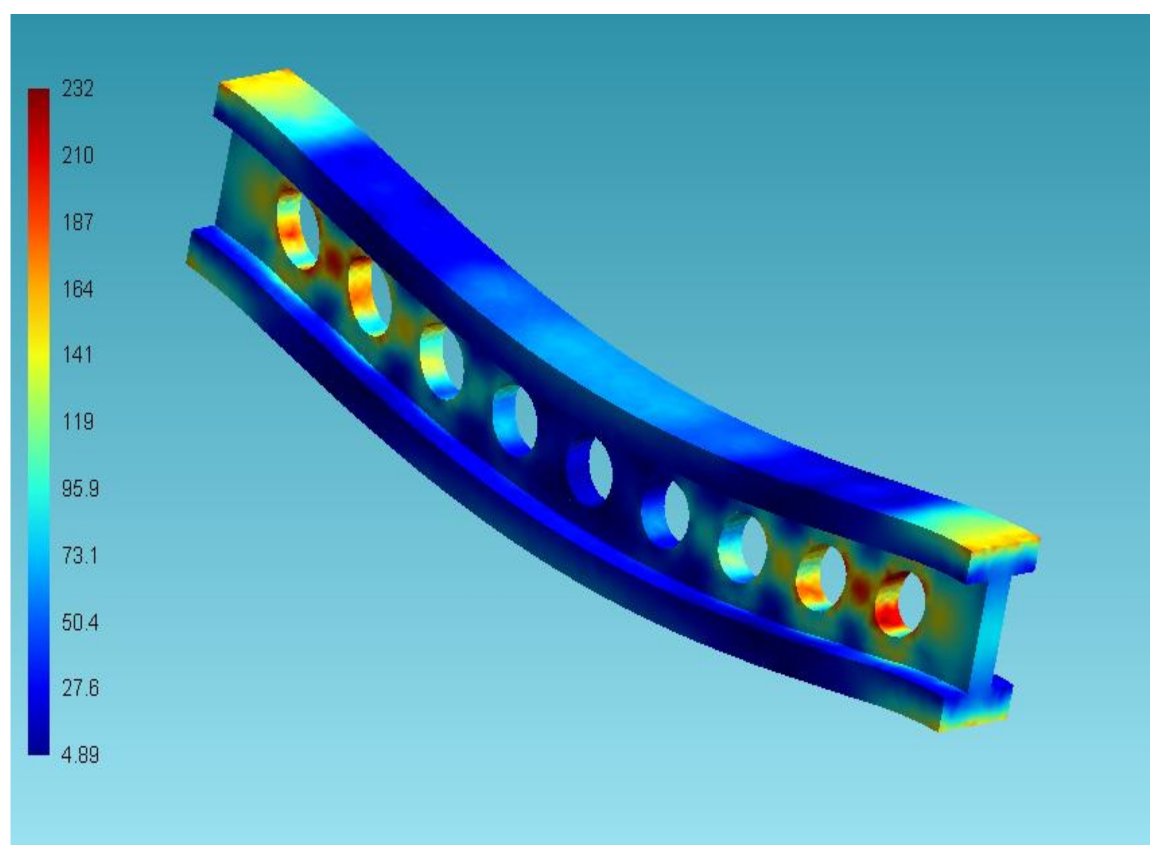


Figure 6.14 : Support beam Mises contours in EAFE.

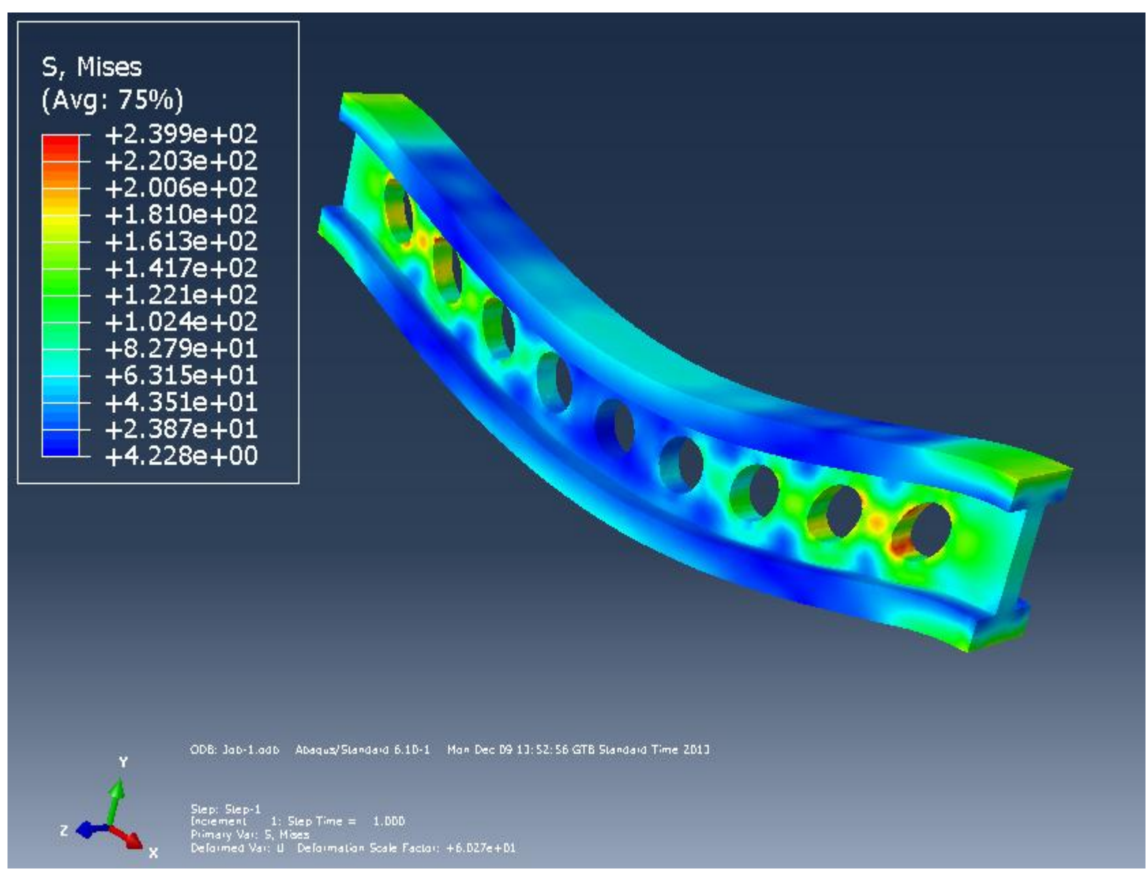


Figure 6.15 : Support beam Mises contours in Abaqus.

A support beam with a uniform pressure example is solved for different element numbers in both EAFE and Abaqus software. The maximum displacement and stress results from six analyses are summarized in Table 6.3.

Table 6.3 : Displacements and stress results with different element numbers.

Number of Elements		Max Displacement (mm)		Max von Mises (MPa)	
EAFE	Abaqus	EAFE	Abaqus	EAFE	Abaqus
134026	133045	0.827	0.829	268	240
92941	97467	0.819	0.822	233	225
54580	55051	0.804	0.807	212	208
17692	17016	0.769	0.754	202	207
10526	10539	0.748	0.724	167	159
4517	4811	0.665	0.677	144	150

6.1.4. Dynamic analysis of a cantilever beam

The problem is graphically represented in Figure 6.16 and EAFE model for this problem is given in Appendix D. The first 8 mode shapes and corresponding natural frequencies obtained from EAFE and Abaqus by using a mesh of 3347 tetrahedral elements are given in Table 6.4 and Table 6.5 respectively. It is shown that the results from EAFE and Abaqus are similar.

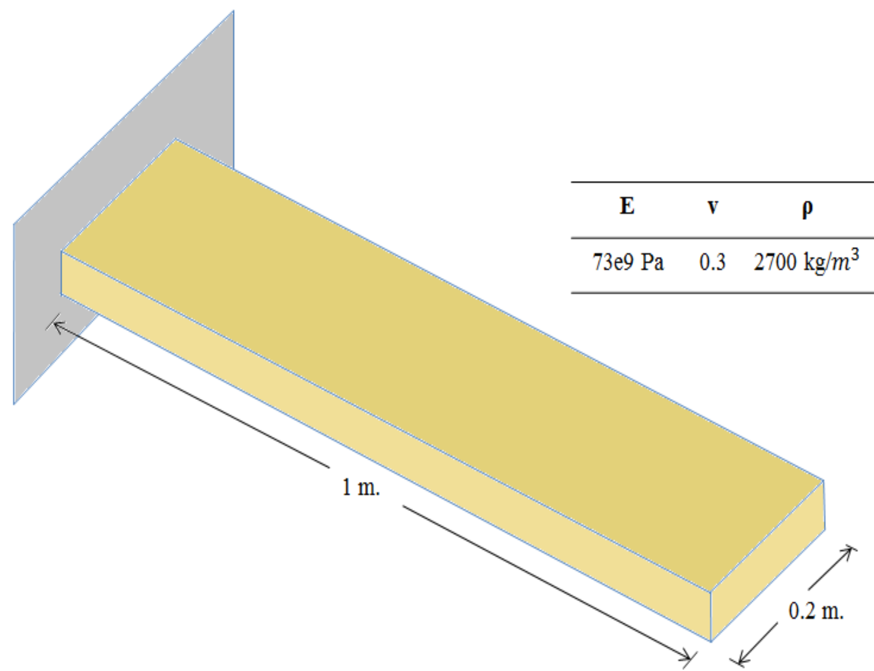
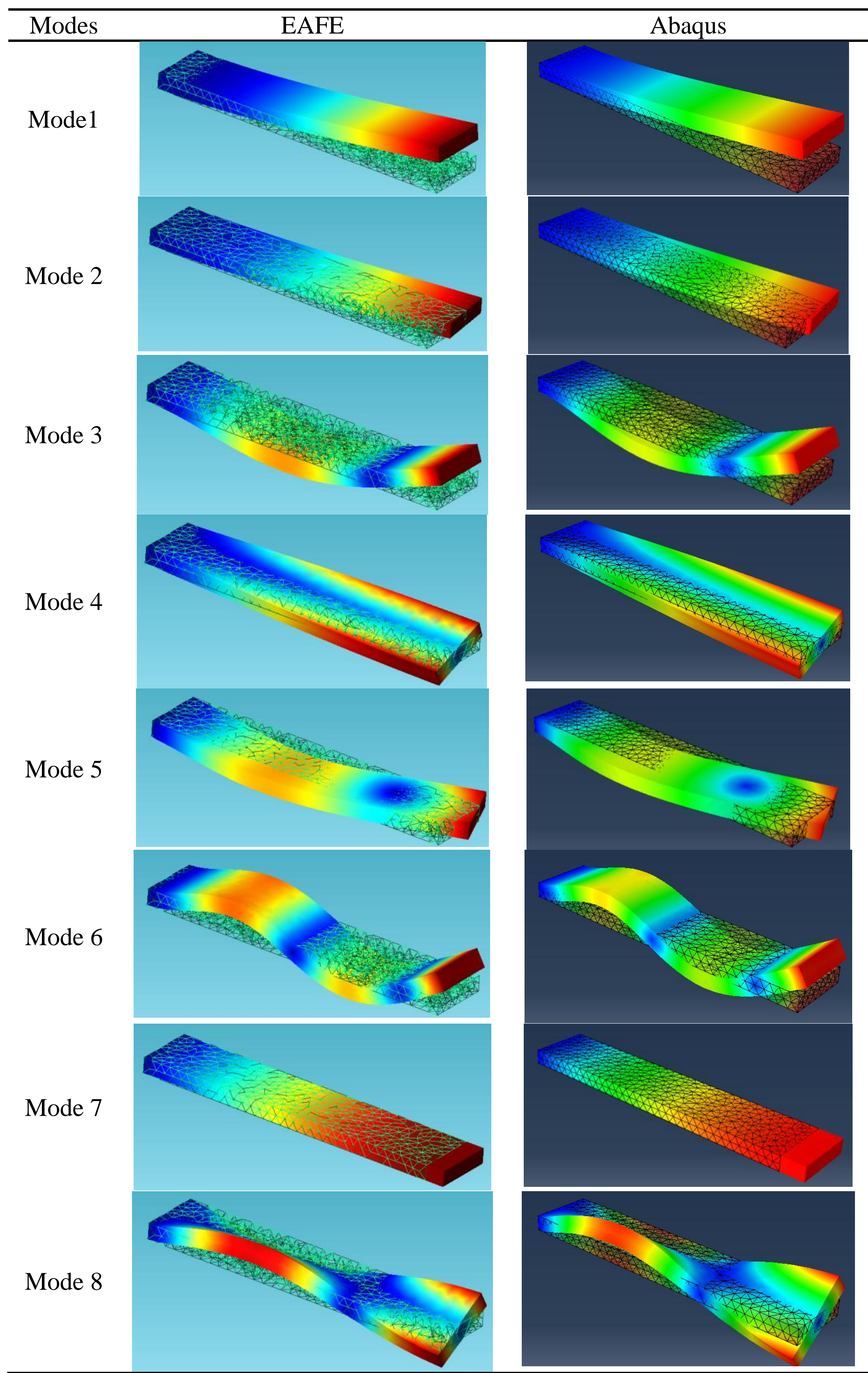


Figure 6.16 : A cantilever beam

Table 6.4 : Cantilever beam mode shapes.



Modes	EAFE	Abaqus
Mode1		
Mode 2		
Mode 3		
Mode 4		
Mode 5		
Mode 6		
Mode 7		
Mode 8		

Table 6.5 : Cantilever beam natural frequencies.

Modes	Natural Frequencies (Hertz)	
	EAFE	Abaqus
1	54.4382	54.412
2	167.1536	167,08
3	335.1094	334.04
4	504.8248	490.5
5	902.6512	900.34
6	920.8526	914.16
7	1306.8326	1306.6
8	1521.8421	1496

7. CONCLUSIONS

In the present work, a 3D finite element software, with a geometry module, a mesh module and a stand-alone solver module is developed by using C++ programming language and an object-oriented programming approach. The geometry module is tested via creating a number of solid parts for finite element analysis, whereas the mesh module is tested by means of their discretization. Moreover, an important aspect of this study, the suitability of object-oriented programming for finite element analysis is demonstrated through developing an FEM processor library: EafeLib. The performance of EafeLib processor is compared with a commercial software package.

The aim at using OOP philosophy in EafeLib processor development was to provide a modular, extensible and reusable FEM software framework. In order to accomplish this, six main classes are used in EafeLib. One of them is the abstract element class which interfaces common virtual functions that any element subclass in this framework must implement. To exemplify usage of these virtual functions a linear tetrahedral element class is developed and in a similar fashion as many different element classes as desired can be easily developed and integrated into the software. Likewise, three other classes: load, boundary condition and material are made abstract to further stimulate extensibility of the software. Another important class in the framework is the model class and it is used to build FEM models. In addition, a node class developed to hold nodal coordinate and displacement data for each node. The flexibility in EafeLib processor is provided by these six base classes.

Three linear static analysis examples are solved both in the developed software and in Abaqus. Nodal displacements and von Mises stress values obtained from the EAFE software are compared to the ones from the Abaqus. It is shown that, the results are almost identical.

The first thing to do to extend EAFE software is to add 1D and 2D modeling capabilities. This can be accomplished by adding 1D and 2D elements such as beam and triangle to the EafeLib processor. Moreover, new solid elements will be added to

let users select an element type from the element library according to the analysis they make.

Currently EAFE software can only handle simple load and boundary condition scenarios. New load and boundary condition types will be added to improve the software.

Mesh module in EAFE software has limited functionalities and is not able to generate meshes with some important element types such as quadrilateral or hexahedral elements. This limitation can be overcome by providing a mesh import function. If a mesh import function is provided, users will be able to use different software for mesh generation.

EAFE software has linear static and linear dynamic analysis options. Different analysis types should be added to the EafeLib processor.

REFERENCES

[1] **Fenves, G. L.** (1990). Object-oriented programming for engineering software development. *Engineering with Computers,* Vol. 6, pp. 1-15.

[2] **Rehak, D. R.** (1986). Artificial intelligence based techniques for finite element program development. In: Bathe K-J, Owen DRJ, editors. *Symposium on Reliability of Methods for Engineering Analysis*. Pineridge Press, pp. 515-532.

[3] **Peskin, R. L., Russo, M. F.** (1988). An object-oriented system environment for partial differential equation solving. *Proceedings ASME Computations in Engineering*, pp. 409-415.

[4] **Miller, G. R.** (1988). A LISP based, object-oriented approach to structural analysis. *Engineering with Computers*, Vol. 4, pp. 197-203.

[5] **Forde, B. W. R., Foschi, R. O., Stiemer, S. F.** (1990). Object-oriented finite element analysis. *Comput Struct*, 34(3), pp. 355–374.

[6] **Duboispelerin, Y. and Zimmermann T.** (1993). Object-Oriented Finite Element Programming.3. An Efficient Implementation in C++. *Computer Methods in Applied Mechanics and Engineering*, Sept., Vol. 108, N1-2, pp. 165-183.

[7] **Duboispelerin, Y., Zimmermann, T., Bomme, P.** (1992). Object-Oriented Finite Element Programming.2. A Prototype Program in Smalltalk. *Computer Methods in Applied Mechanics and Engineering*, Aug., Vol. 98, N3, pp. 361-397.

[8] **Menetrey, P., Zimmermann, T.** (1993). Object-Oriented Non-Linear Finite Element Analysis - Application to J2 Plasticity. *Computers & Structures*, Dec. 3, Vol. 49, N5, pp. 767-777.

[9] **Zimmermann, T., Duboispelerin, Y., Bomme, P.** (1992). Object-Oriented Finite Element Programming .1. Governing Principles, Computer Methods in Applied Mechanics and Engineering, Jul., Vol. 98, N2, pp. 291-303.

[10] **Lu, J., White, D., Chen, W. F.** (1993). Applying object-oriented design to finite element programming. *SAC '93: proceedings of the 1993 ACM/SIGAPP symposium on applied computing*. ACM, New York, pp. 424–429.

[11] **Lu, J., White, D. W., Chen, W. F., Dunsmore, H. E.** (1995). A matrix class library in C++ for structural engineering computing. *Comput Struct*. Vol. 55, N1, pp. 95–111.

[12] **Bangerth, W., Hartmann, R., Kanschat, G.** (2006). deal.II—a general purpose object oriented finite element library. Technical Report. ISC-06-02-MATH, Institute for Scientific Computation, Texas A&M University.

[13] **Bangerth, W., Hartmann, R., Kanschat, G.** deal.II Differential equations analysis library, Technical reference. Date retrieved: 17.10.2013, address: *http://www.dealii.org*

[14] **Patzák, B., Bittnar, Z.** (1999). Object oriented finite element modeling. *Acta Polytech* 39(2), pp. 99–113.

[15] **Holmevik, J., R.** Compiling SIMULA: A historical study of technological genesis.

[16] **Url-1** <*http://www.ntecs.de/old-hp/uu9r/lang/html/uml_shape.png*>, date retrieved 08.09.2013.

[17] **UML**, Superstructure Specification Version 2.2. OMG, February 2009.

[18] **Forde, B. W. R., Foschi, R. O. and Steimer, S. F.** (1990). Object-oriented finite element analysis. *Computers and Structures*, Vol. 34, pp. 355–374.

[19] **Url-2** <*http://homepage.usask.ca/~ijm451/finite/fe_resources/node139.html*>, date retrieved 07.07.2013.

[20] **Deitel, H. M., Deitel, P. J.** (2003). *Java, How to Program*, Deitel Associates Inc. 5th edition, Prentice Hall, USA, pp. 1-25.

[21] **Deitel, P., Deitel H.** (2012). *C++ How to Program*, Deitel Associates Inc. 8th edition, Prentice Hall, USA, pp. 17-18.

[22] **Hortons, I.** (2010). *Beginning Visual C*, Wiley, USA, pp. 877.

[23] **Guha, S.** (2011). *Computer graphics through OpenGL from theory to experiments*, CRC Press, USA, pp. 6.

[24] **Shirley**, **P., Marschner**, **S**. (2009). *Fundamentals of computer graphics*, 3rd edition, CRC Press, USA, pp. 3.

[25] **Sarcar, M. M. M., Rao, M. K., Narayan, L. K.** (2008), Computer aided design and manufacturing , pp. 3.

[26] **Url-3** <*http://design.osu.edu/carlson/history/lesson10.html*>, date retrieved 06.05.2013.

[27] **Richard, S., Wright, Jr., Haemel, N., Sellers, G., Lipchak, B.** (2011). *OpenGL Super Bible Comprehensive Tutorial and Reference*, 5th edition, Addison-Wesley, USA, pp. 11–18.

[28] **Wireframe teapot.** (n.d.). Date retrieved: 06.07.2013, address: *http://caig.cs.nctu.edu.tw/course/CG2007*

[29] **Modern render.** (n.d.). Date retrieved: 06.07.2013, address: *http://commons.wikimedia.org/wiki/File:Utah_teapot_simple_2.png*

[30] **Original teapot.** (n.d.) Date retrieved: 06.07.2013, address: *http://boakes.org/2005/07/01/teapot/*

[31] **Guha, S.** (2011). *Computer Graphics through OpenGL, from theory to experiments*, CRC Press, USA, pp. 35-39.

[32] **Url-4** <*http://math.hws.edu/graphicsnotes/c2/s4.html*>, date retrieved 07.04.2013.

[33] **Eck., D. J.** Fundamentals of computer graphics with java, openGL and jogl, pp. 51-54.

[34] **Url-5** <*http://gfxspeak.com/2013/06/06/does-the-cad-world-need-another-geometry-kernel/*>, date retrieved 07.05.2013.

[35] **OCCT Object Libraries**, Application Framework user guide, version 6.5.5, March 2013.

[36] **Url-6** <*http://www.salome-platform.org/*>, date retrieved 21.08.2013.

[37] **Geuzaine, C., Remacle, J. F.** (2013), *Gmsh reference manual* , pp. 5-6.

[38] **Geuzaine, C. and Remacle, J. F.** (2009). Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, Vol. 79, N11, pp. 1309-1331.

[39] **Fish, J., Belytschko, T.** (2007). *A First Course in Finite Elements*, Wiley, USA, pp. 1-2.

[40] **Chandrupatla, T. R., Belegundu, A. D.** (2002). *Introduction to finite elements in Engineering*, 3rd edition, Prentice Hall, pp. 275-284.

[41] **Url-7** <*http://www.engapplets.vt.edu/Mohr/java/nsfapplets/MohrCircles2 3D/Theory/brick.gif*>, date retrieved 08.05.2013.

[42] **Url-8** < *http://www.colorado.edu/engineering/CAS/courses.d/AFEM.d/AFE M.Ch09.d/AFEM.Ch09.pdf*>, date retrieved 23.10.2013.

[43] **Lakshmininarayana, H.** (2004). *Finite Elements Analysis: Procedures in Engineering*, Universities Press, pp. 111-112.

[44] **Balay, S., Brown, J., Buschelman, K., Eijkhout, V., Gropp, W., Kaushik, D., Knepley, M., Curfman L., McInnes, Smith, B. and Zhang H.** (2013). PETSc Users Manual, Revision 3.4 by Mathematics and Computer Science Division, Argonne National Laboratory, pp. 19-21.

[45] **Sala, M., Heroux, M. A., Day, D. M., Willenbring J. M.** (2004). *Trilinos user manual*, Trilinos Release 10.12, Sandia Report SAND2004-2189, pp. 1-8.

APPENDICES

APPENDIX A: EAFE model for cantilever beam example

APPENDIX B: EAFE model for a plate with hole example

APPENDIX C: EAFE model for support beam example

APPENDIX D: EAFE model for modal analysis of a cantilever beam

APPENDIX A

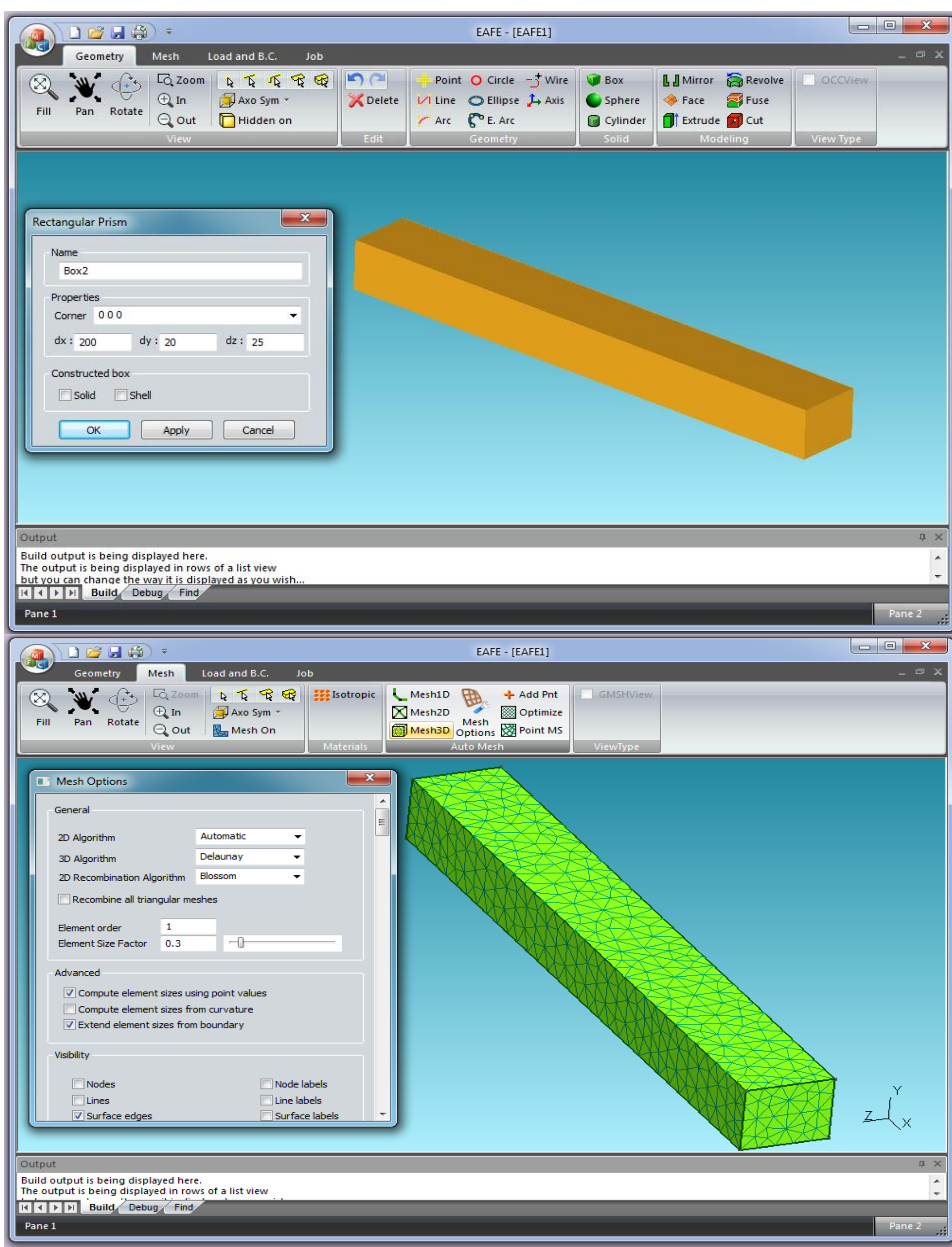


Figure A.1 : Solid model of cantilever beam and its finite element mesh.

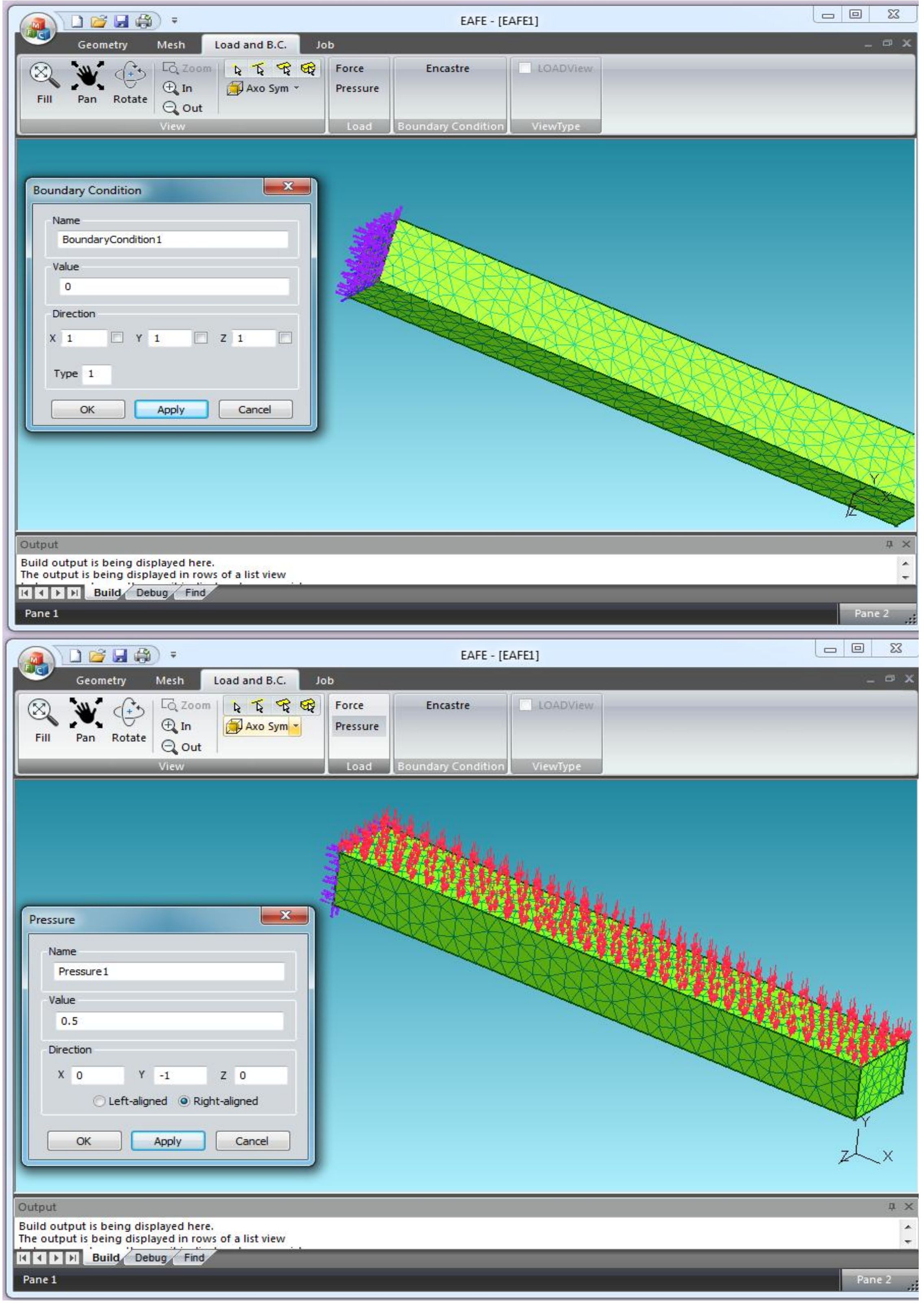


Figure A.2 : Defining boundary condition and load for the model.

APPENDIX B

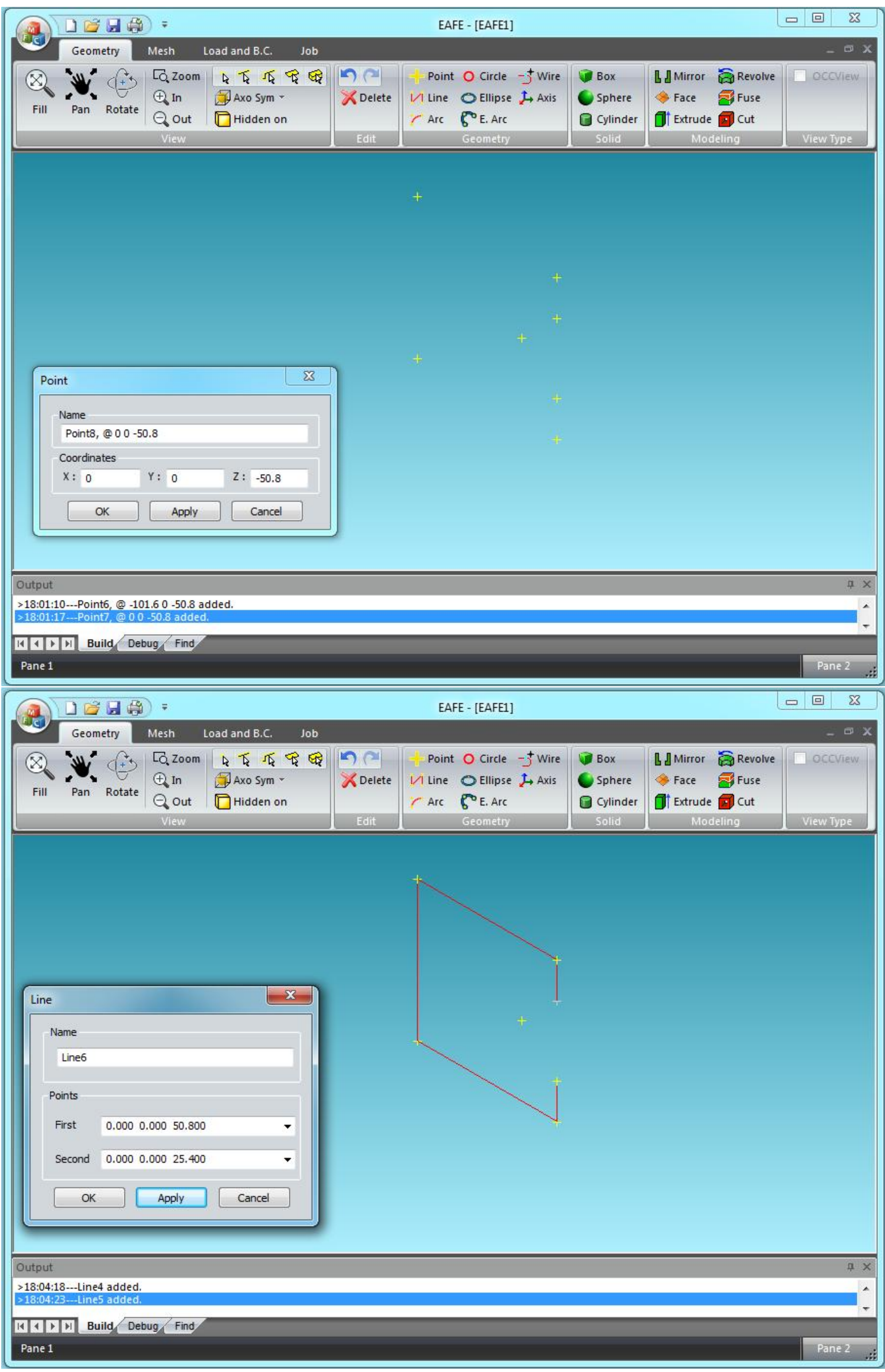


Figure B.1 : Adding points and lines to create half of the plate.

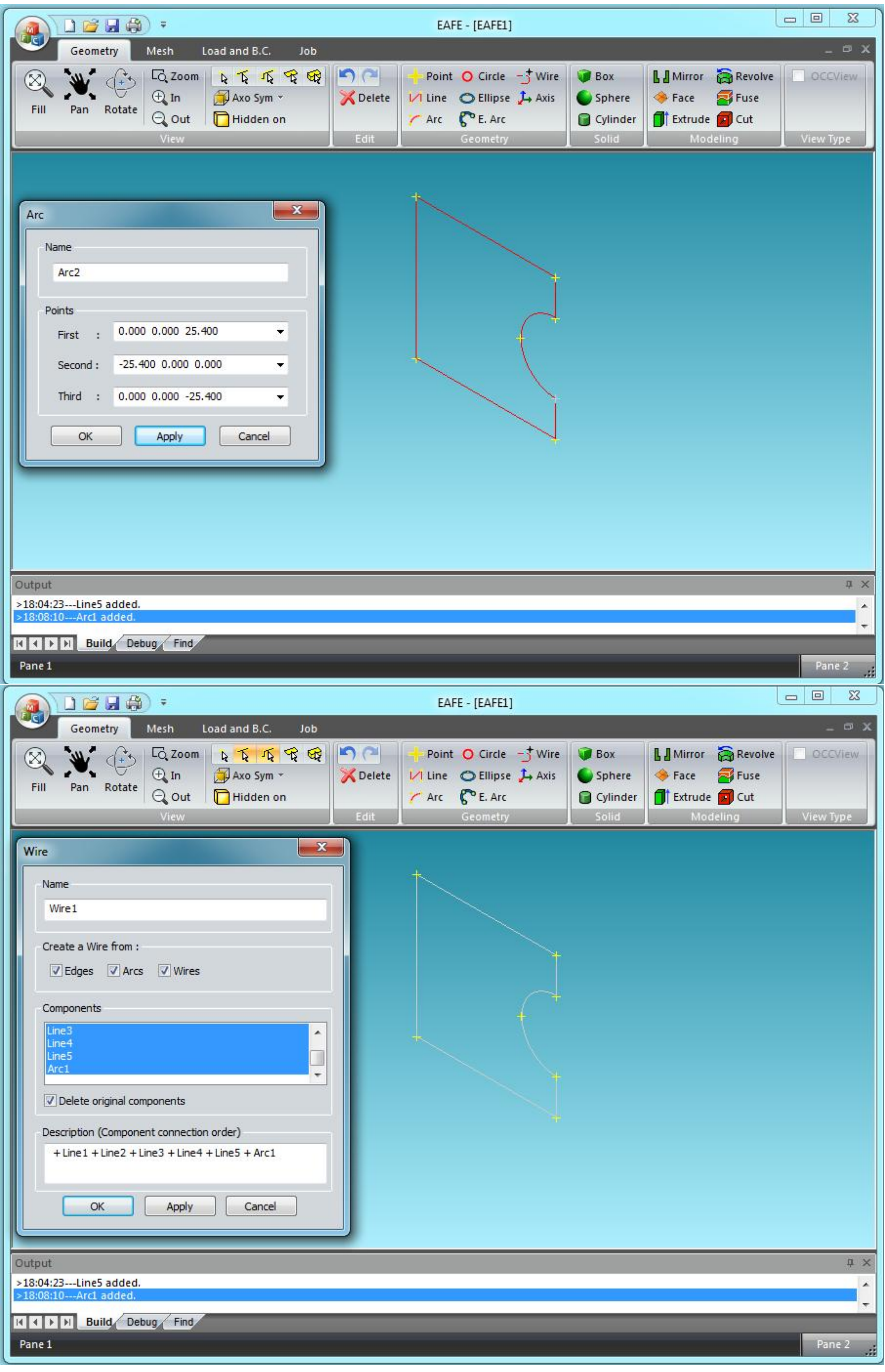


Figure B.2 : Creating a wire by connecting consecutive lines and an arc.

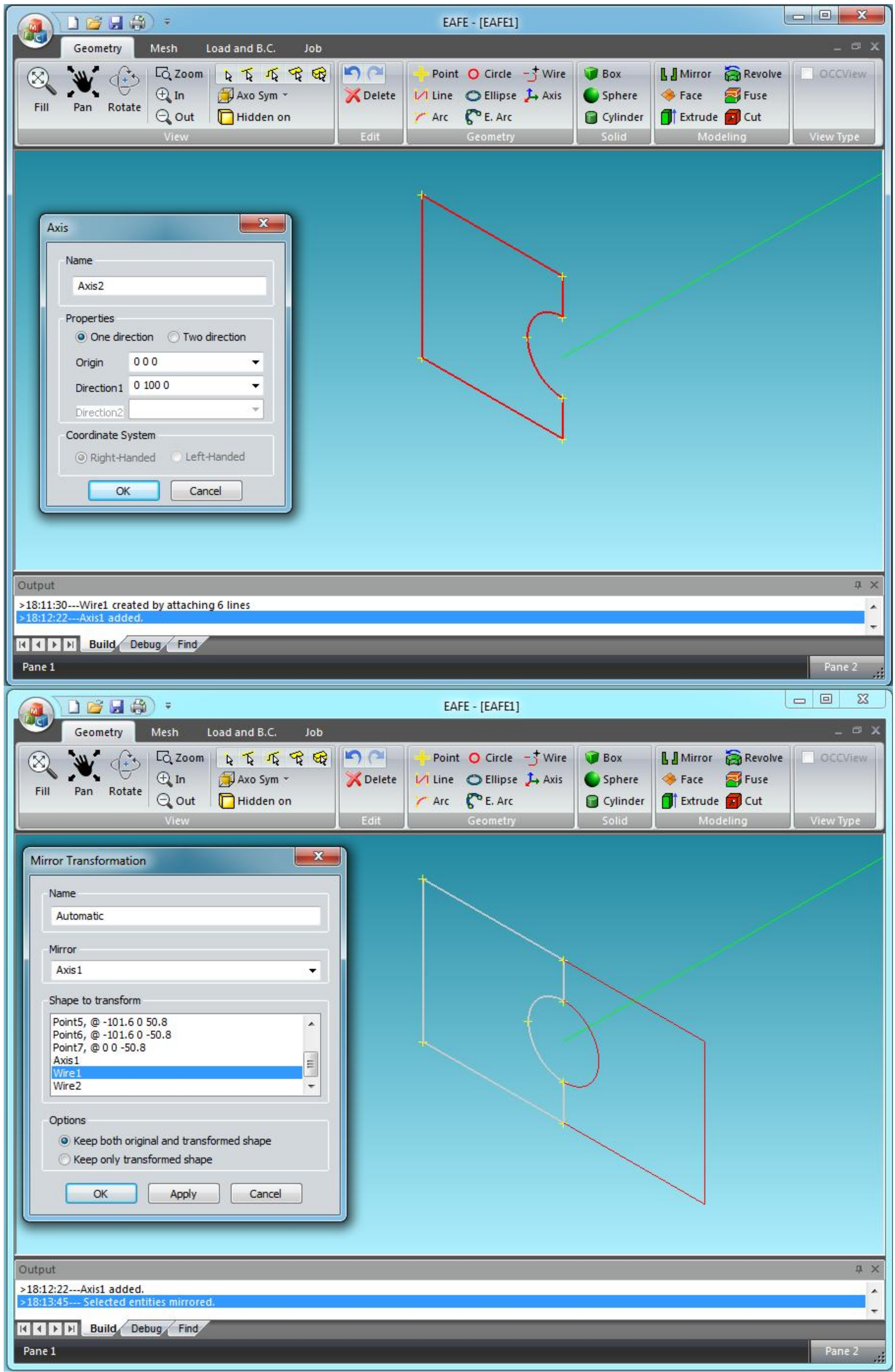


Figure B.3 : Using mirror function with an axis to complete the frame of the plate.

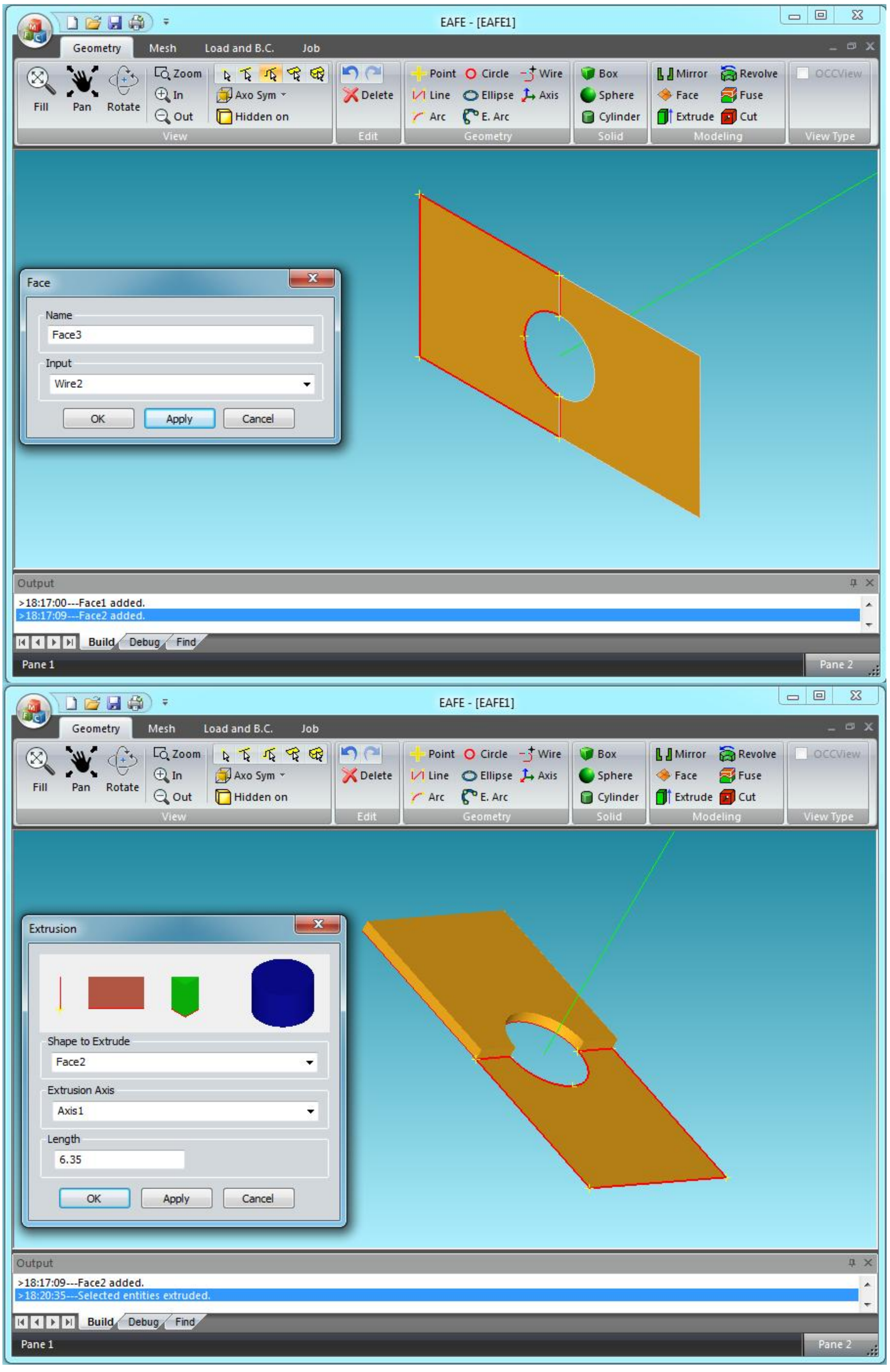


Figure B.4 : Creating two faces and extruding them along an axis.

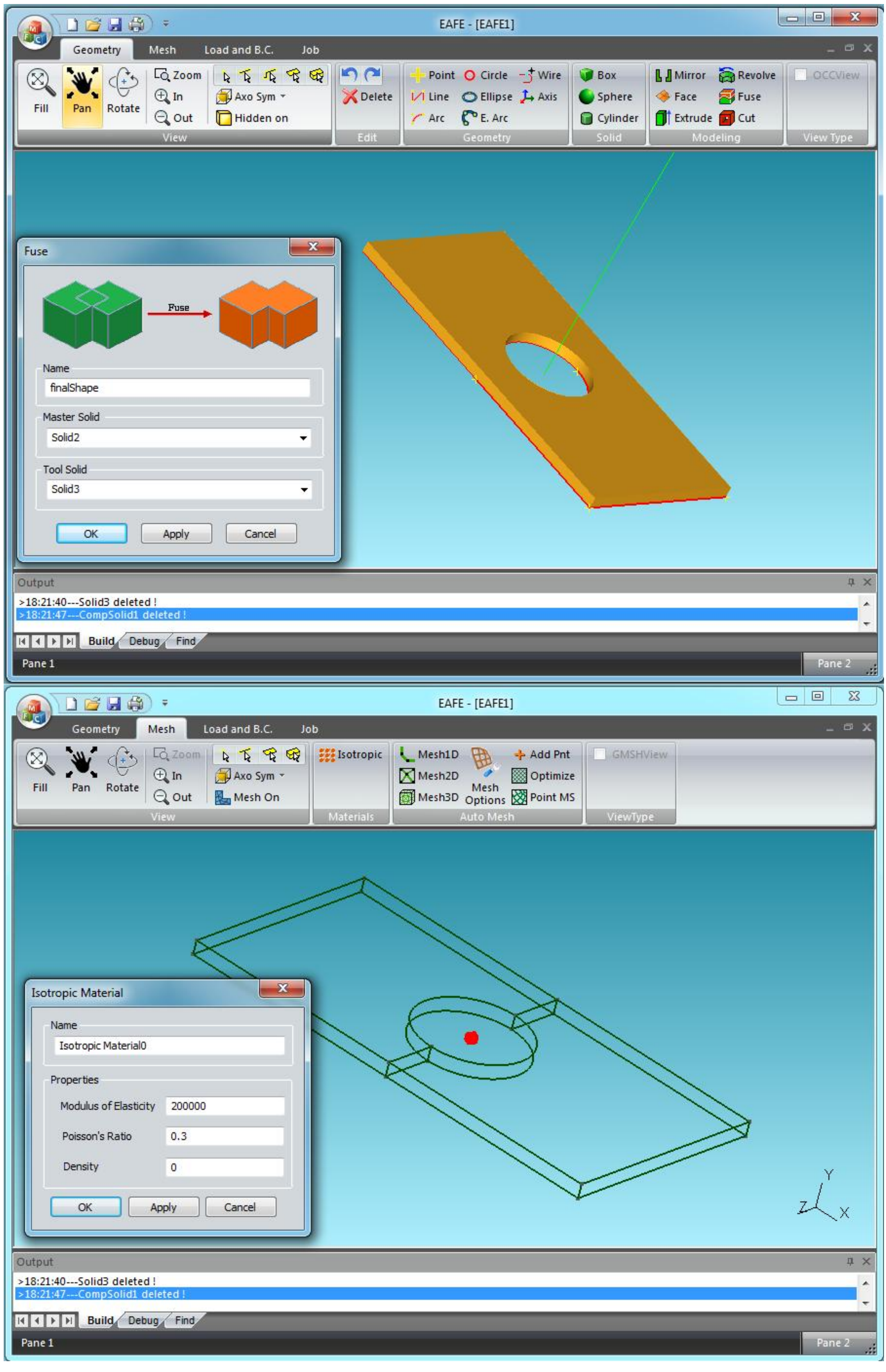


Figure B.5 : Fusing two separate halves and defining material for the plate.

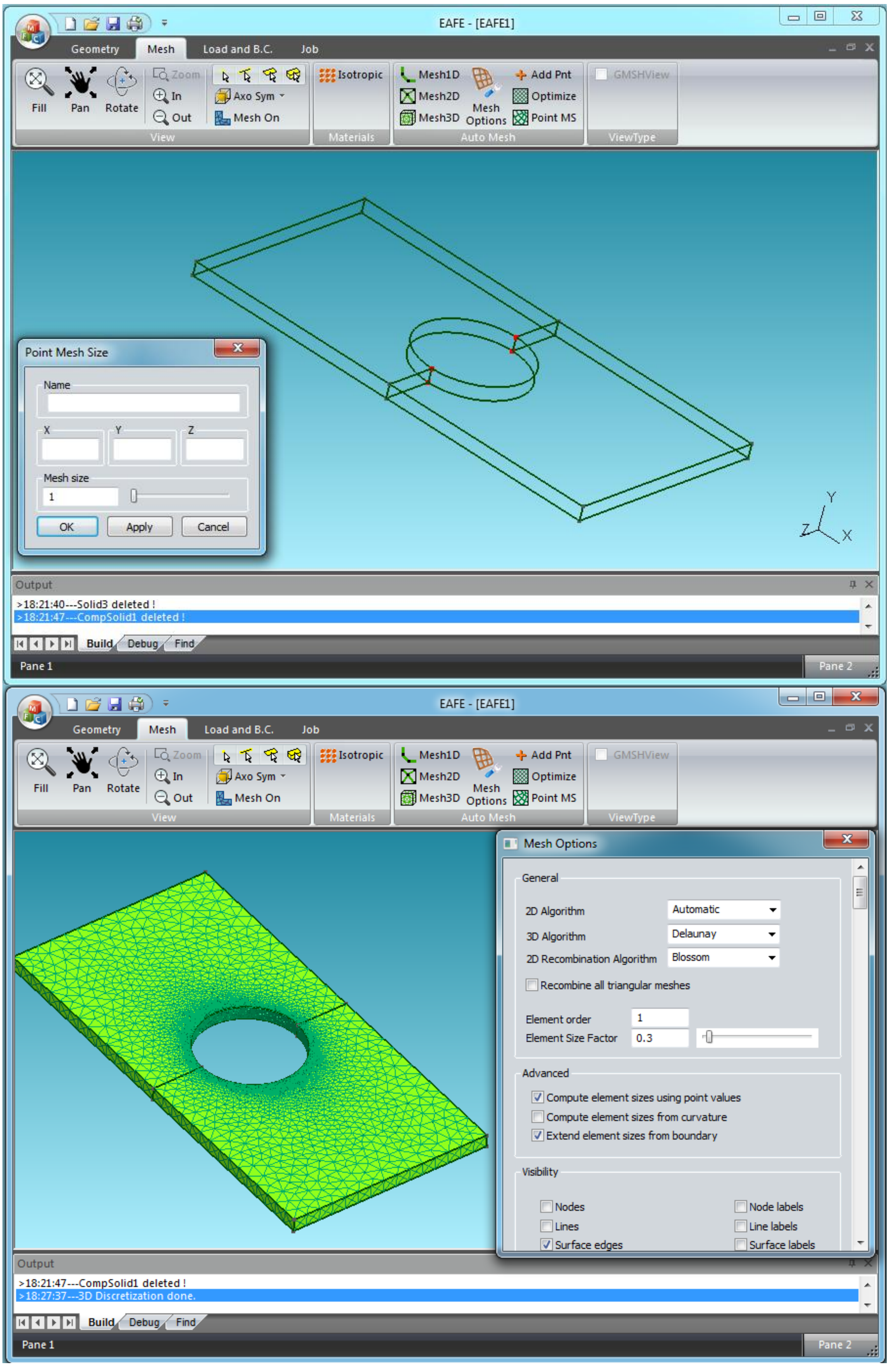


Figure B.6 : Discretizing the plate by increasing mesh density on critical region.

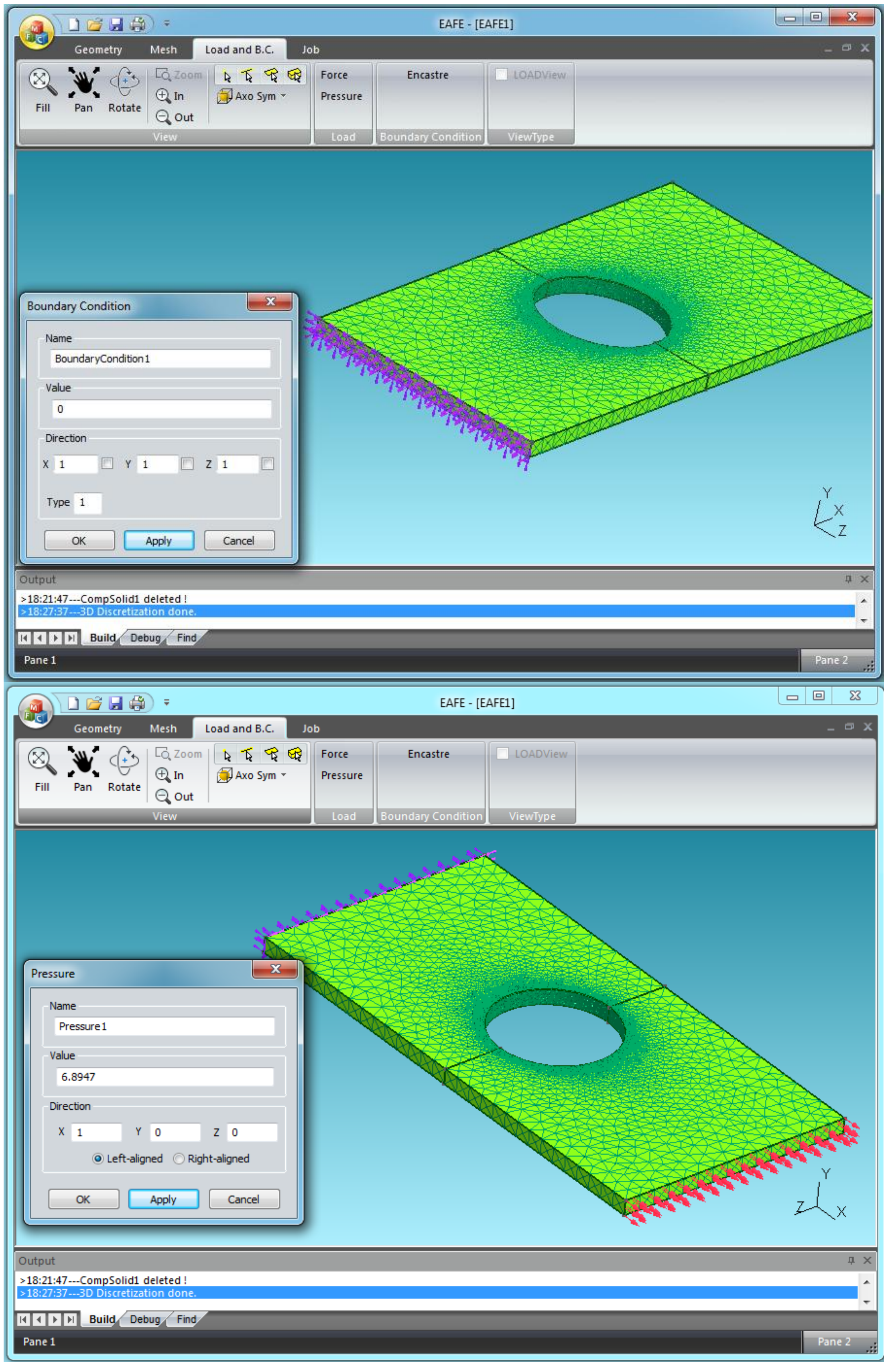


Figure B.7 : Defining boundary condition and load for the model.

APPENDIX C

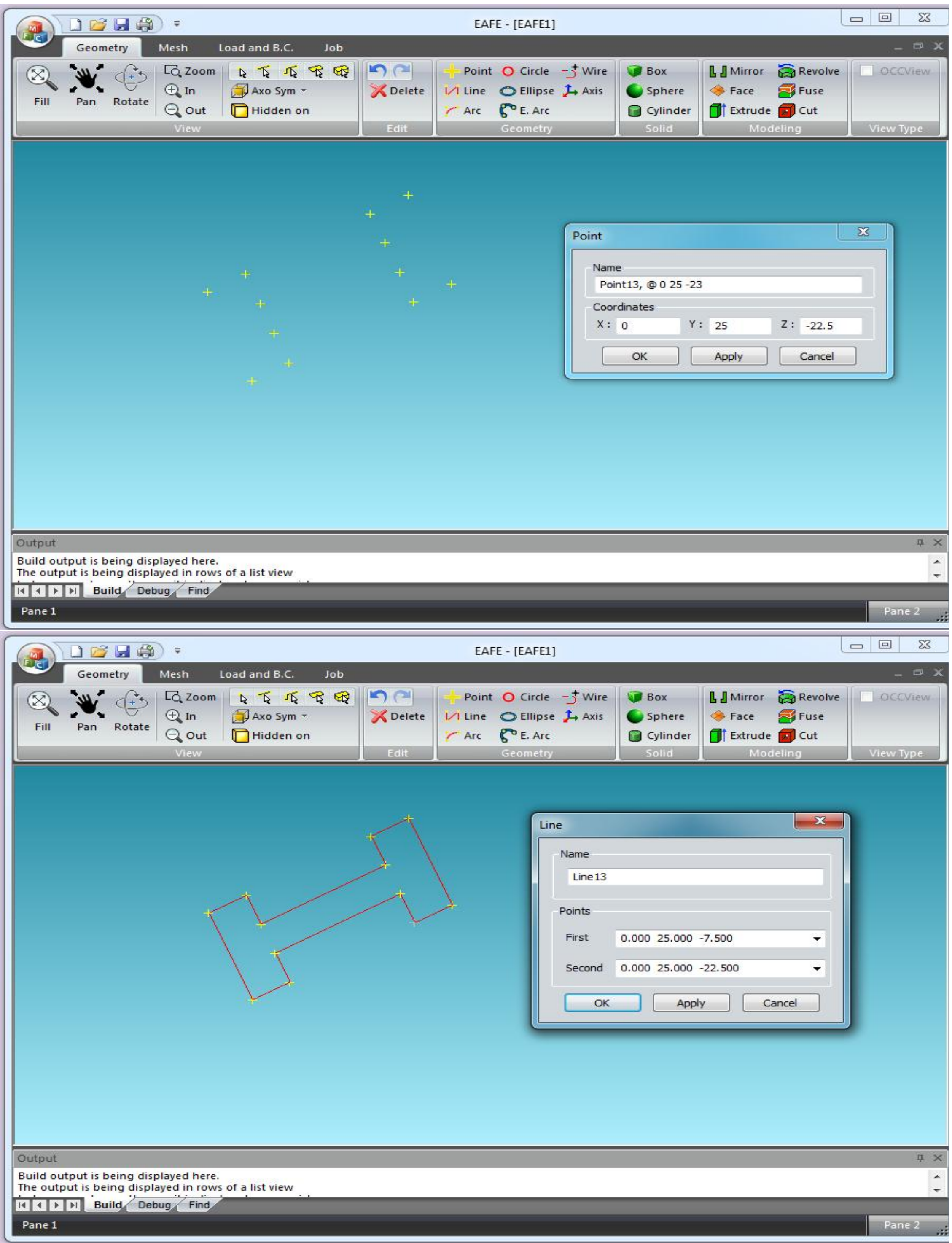


Figure C.1 : Adding points and lines to create the cross-section of the model.

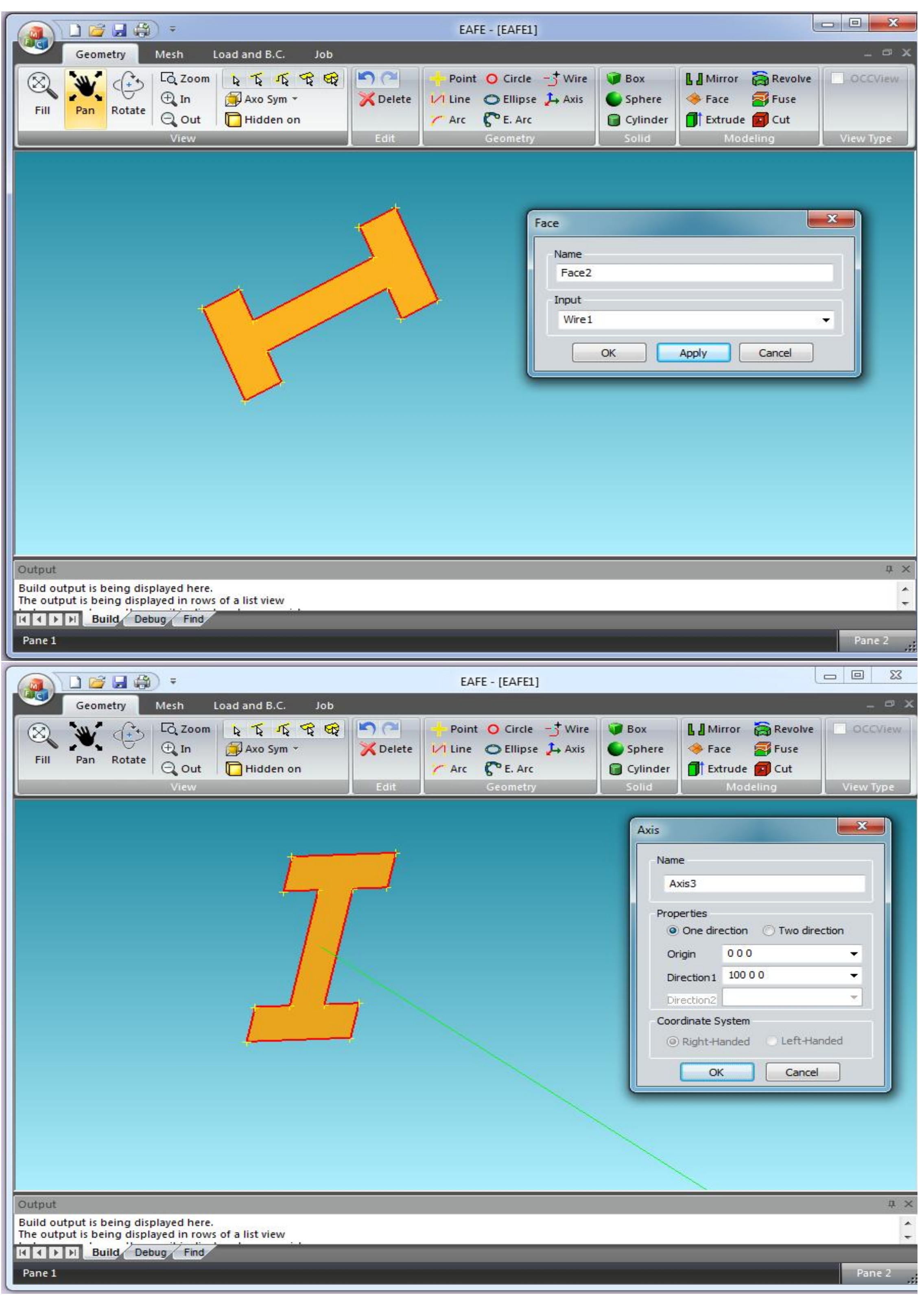


Figure C.2 : Creating a face and an axis for extrusion.

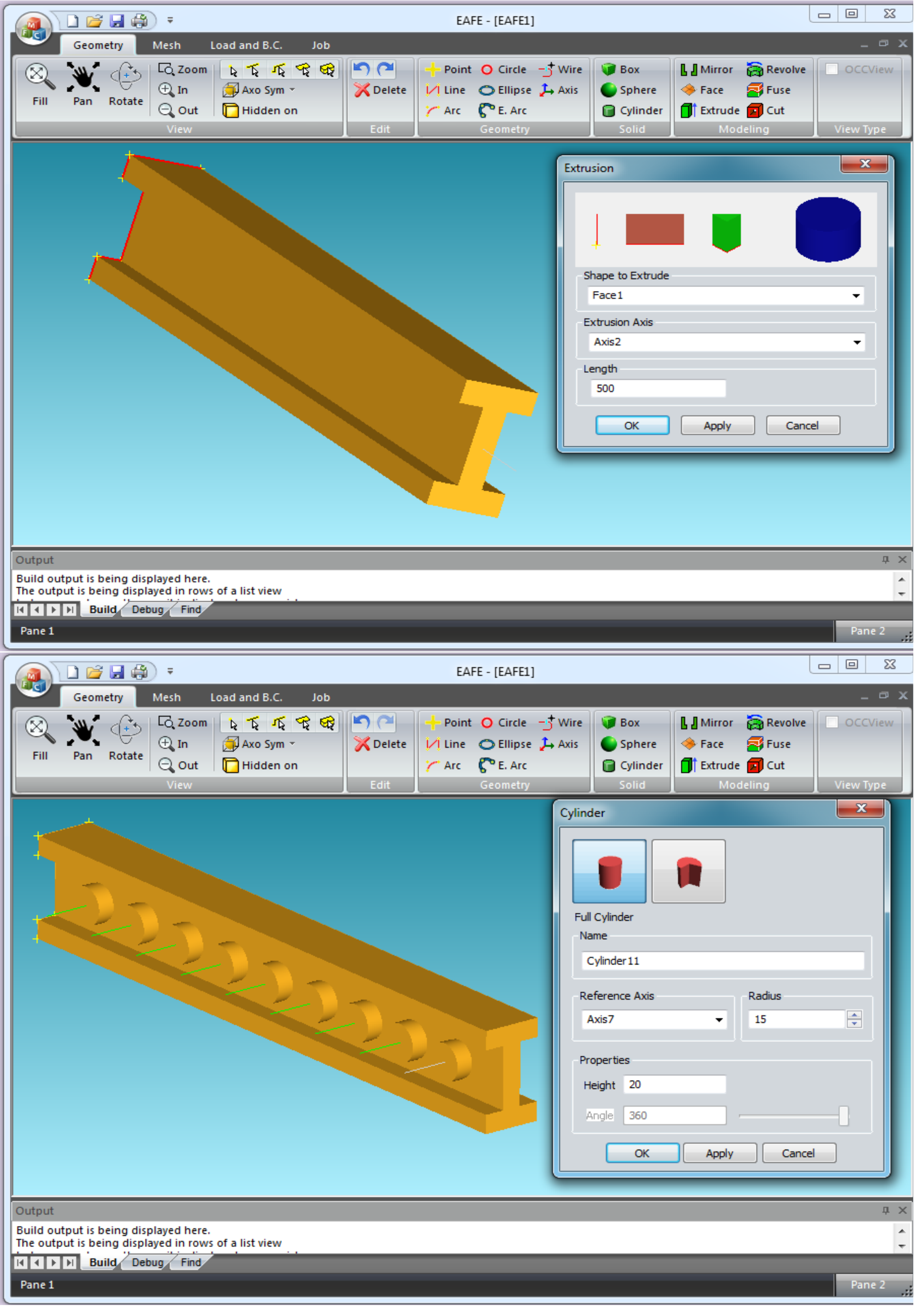


Figure C.3 : Extruding the face along the axis and creating cylinders for the holes.

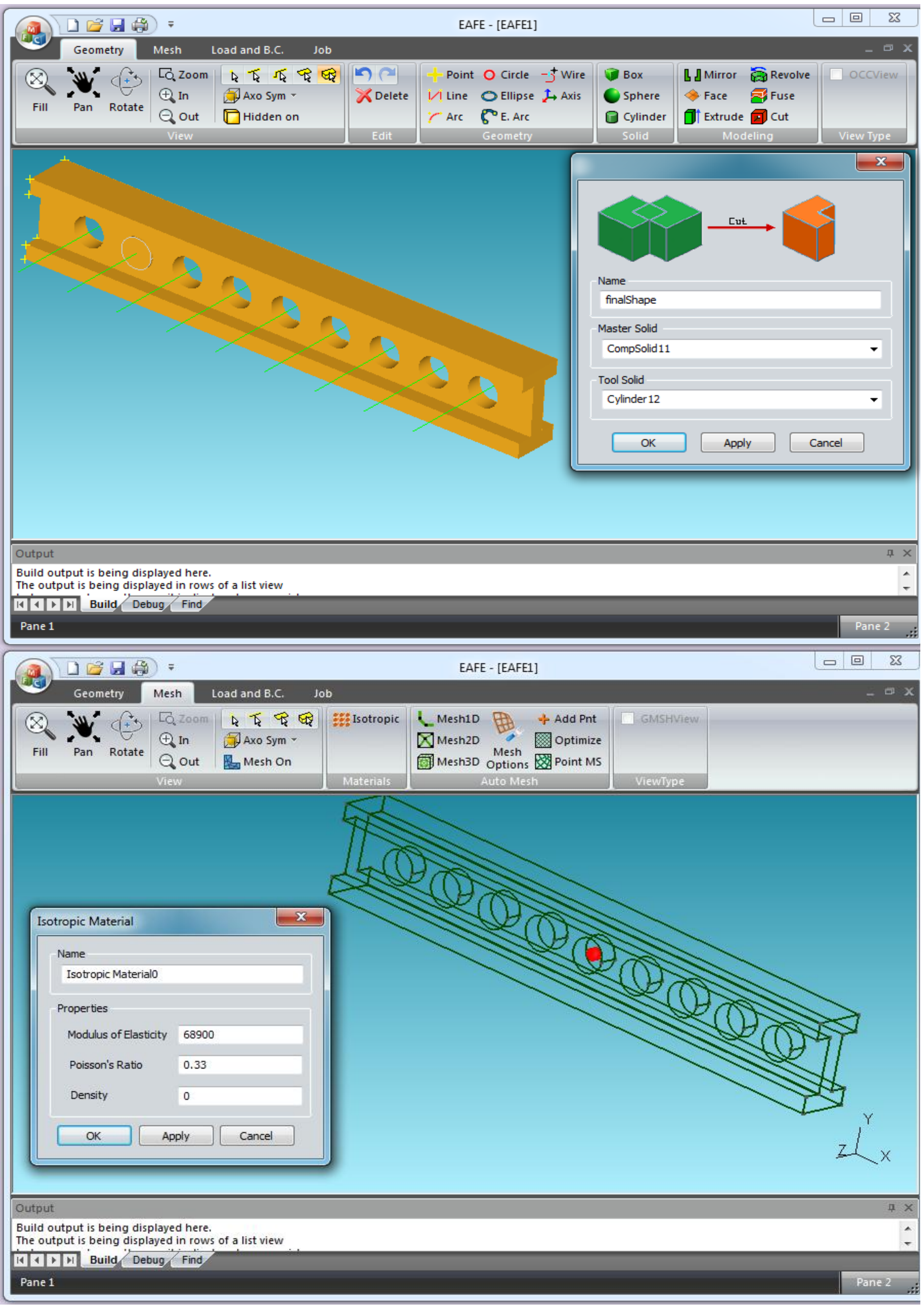


Figure C.4 : Cutting cylinders from the part and defining material for the model.

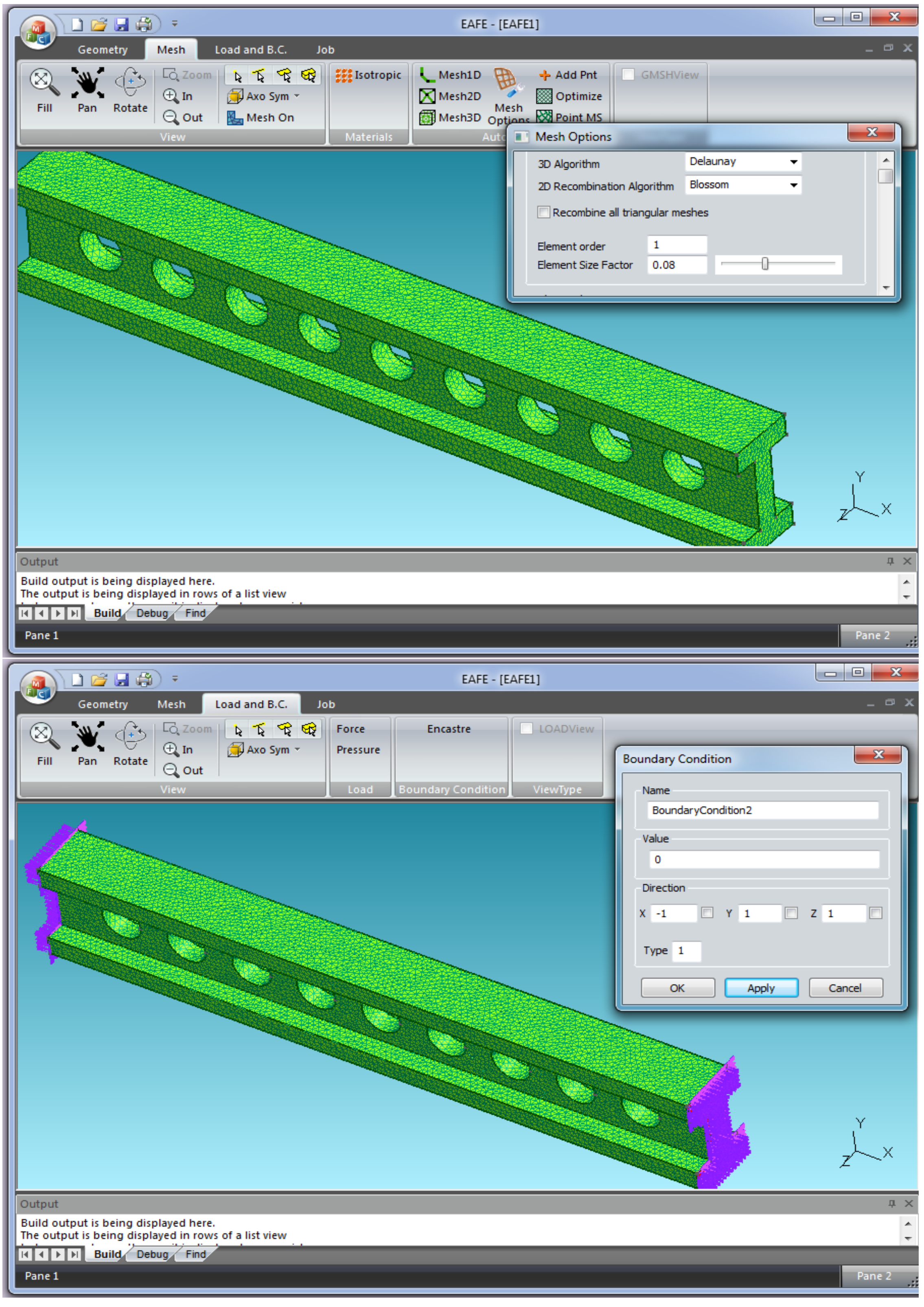


Figure C.5 : Discretizing the model and adding boundary conditions.

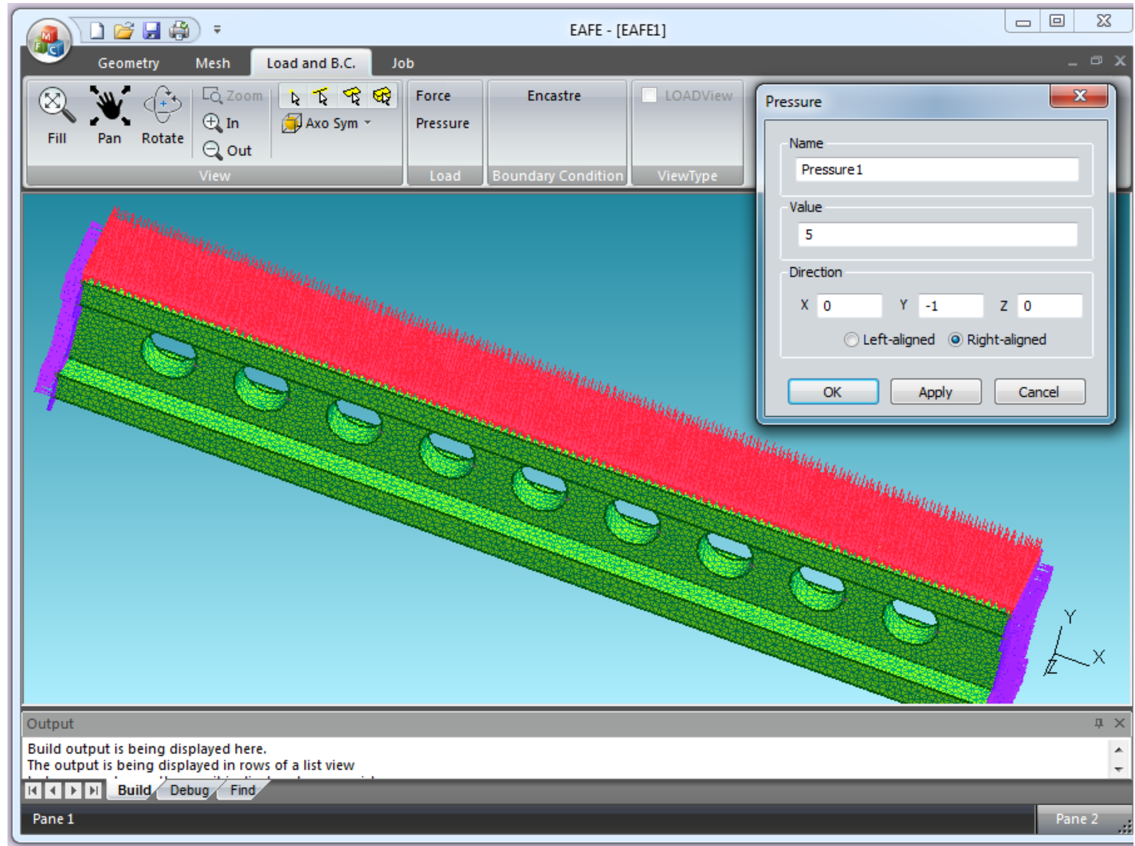


Figure C.6 : Adding distributed load.

APPENDIX D

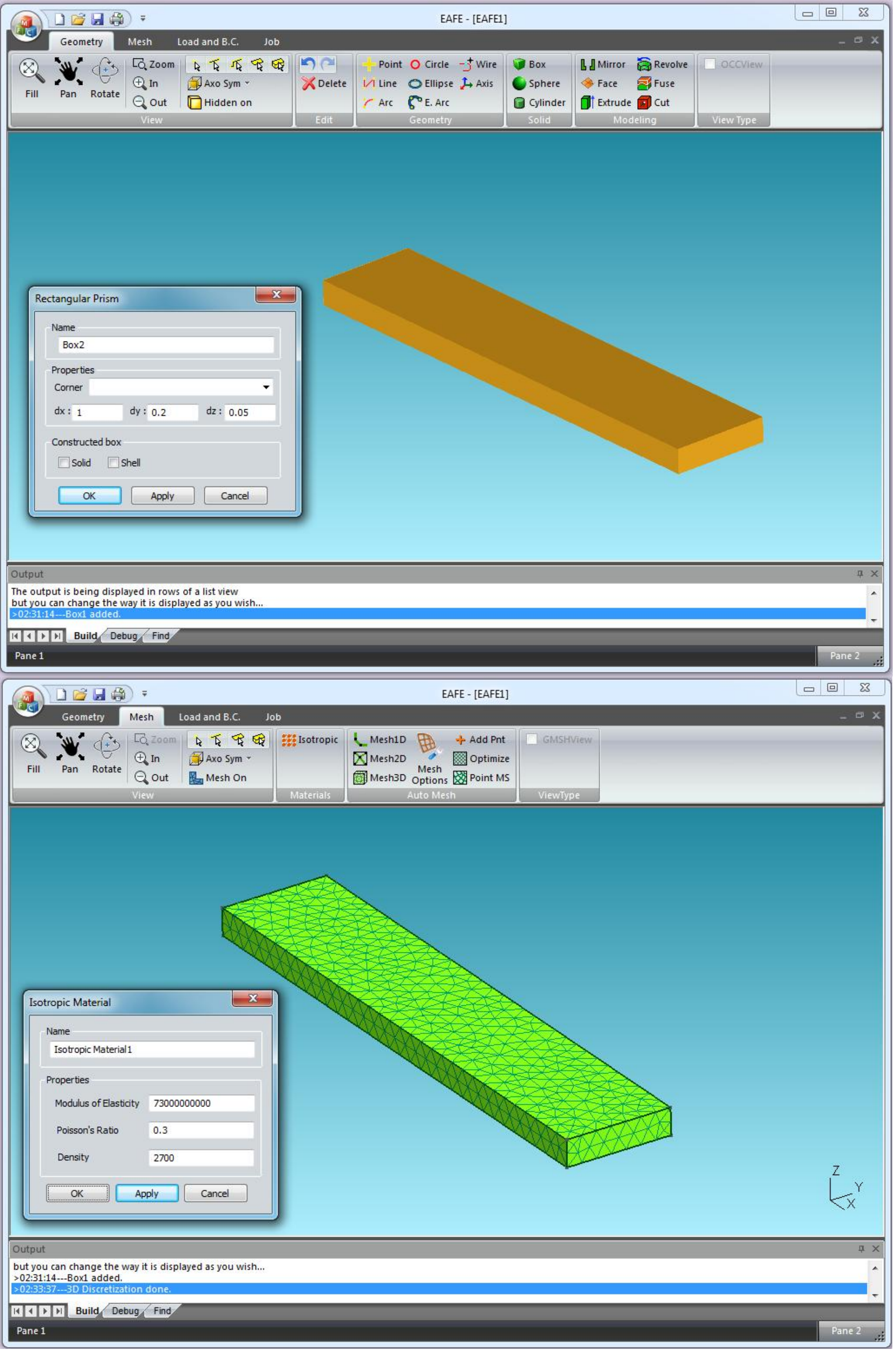


Figure D.1 : Solid model of cantilever beam and its finite element mesh.

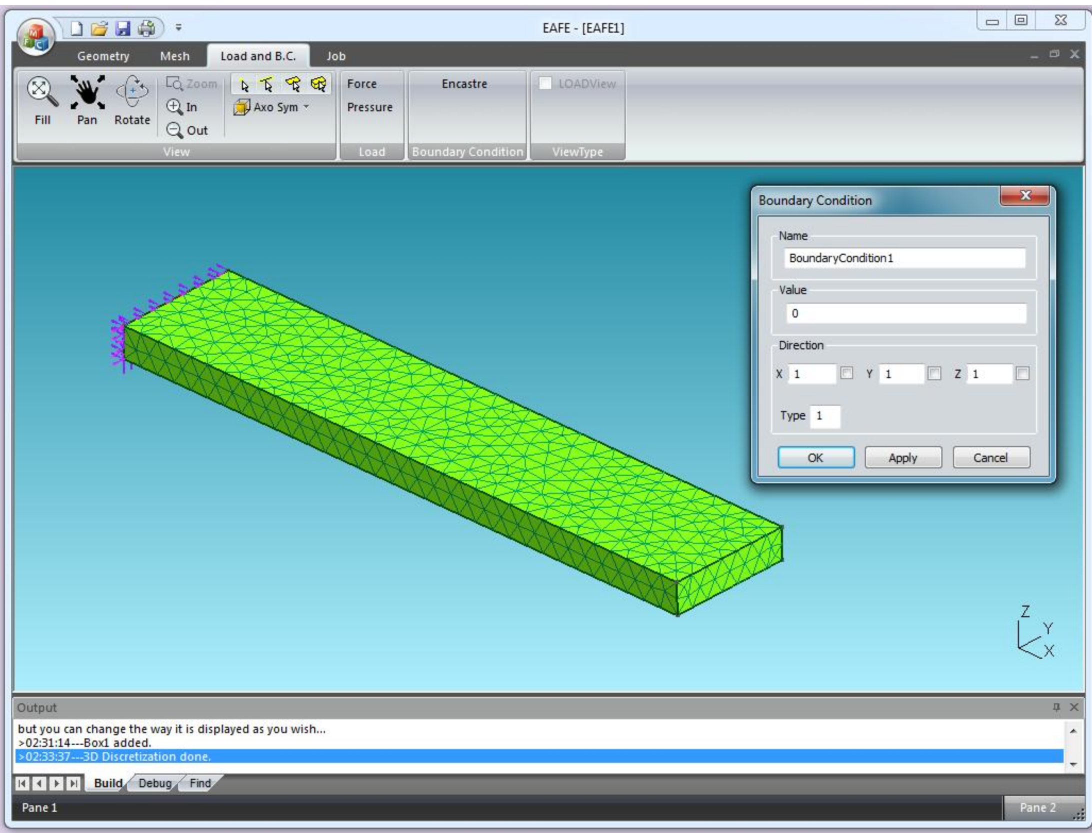


Figure D.2 : Defining boundary condition for the model.

CURRICULUM VITAE

Name Surname: Halid Eren Adak

Place and Date of Birth: İstanbul, 23/10/1987

E-Mail: erad_5@hotmail.com

BSc: Industrial Engineering (Kocaeli University)