Programming the Finite Element Method using C# and the .NET Framework

Serdar Astarlioglu

Objective

- To provide an introduction to programming the FEM using object-oriented approach with C# and the .NET Framework

Outline

- The .NET Framework
  - The .NET Framework
  - C# Programming Language
- Procedural FEM
  - Data structures
  - Subroutines
- Object-Oriented FEM
  - Classes
- Discussion

The .NET Framework

- Consists of
  - Common Language Runtime (CLR)
  - Base Class Libraries (BCL)
- Object-oriented platform
- Supports a wide range of programming languages
- Code targeting the CLR is called “managed code”

C# Programming Language

- Member of C family of languages (C/C++/Java)
- Short learning curve
- Everything is an object
- Code written “in-line”
- Supports integrated XML documentation
- Supports Unicode

C# Programming Language

```
using System;

namespace Snaps3D.Algebra
{
   [Serializable]  public class DoubleVector
   {
      private int _length;

      public int Length
      {
         get { return _length; }
      }

      public double GetNorm()
      {
         // Initialize norm
         double normSq = 0;

         for (int i = 0; i < Length; i++)
            normSq += this[i] * this[i];
         return Math.Sqrt(normSq);
      }
   }
}
```

- Namespaces
- Types
  - Classes, structs, interfaces, enums, and delegates
  - Value type or reference type
- Members
  - Fields, properties, methods, operators, attributes, etc.

Outline

- The .NET Framework
  - The .NET Framework
  - C# Programming Language
- Procedural FEM
  - Data structures
  - Subroutines
- Object-Oriented FEM
  - Classes
- Discussion

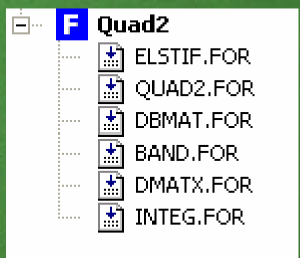
Procedural FEM

- Program contains multiple procedures
- Each procedure implements a specific algorithm
- Procedures use/manipulate data
- Data is numbers and text
- Code crunches data

Data Structures

```
C    *****                      QUAD2                        *****
C    ***      2-D STRESS ANALYSIS USING 4-NODE                 ***
C    ***                QUADRILATERAL ELEMENTS                 ***
      DIMENSION X(100,2),NOC(100,4),MAT(100),PM(10,3)
      DIMENSION NU(50),U(50),S(200,50),F(200),D(3,3),TH(100)
      DIMENSION B(3,8),DB(3,8),SE(8,8),Q(8),STR(3),TL(8)
      DIMENSION DT(100),XNI(4,2),MPC(20,2),BT(20,3)
      CHARACTER*16 FILE1,FILE2,FILE3
      CHARACTER*81 DUMMY,TITLE
      IMAX = 200
```

Subroutines



SUBROUTINE INTEG(XNI)
Returns the integration point coordinates, XNI (4x2)

SUBROUTINE DMATX(N,PM,MAT,PNU,AL,LC,D)
Returns the material matrix, D (3x3)

SUBROUTINE ELSTIF(N,LC,SE,TL,XNI,D,TH,DT,X,NOC,AL,PNU)
Returns the element stiffness matrix, SE (8x8)

SUBROUTINE DBMAT(N,X,NOC,TH,THICK,D,B,DB,DJ,XI,ETA)
Returns the matrix relating stresses to nodal displacements at an integration point, DB (3x8)

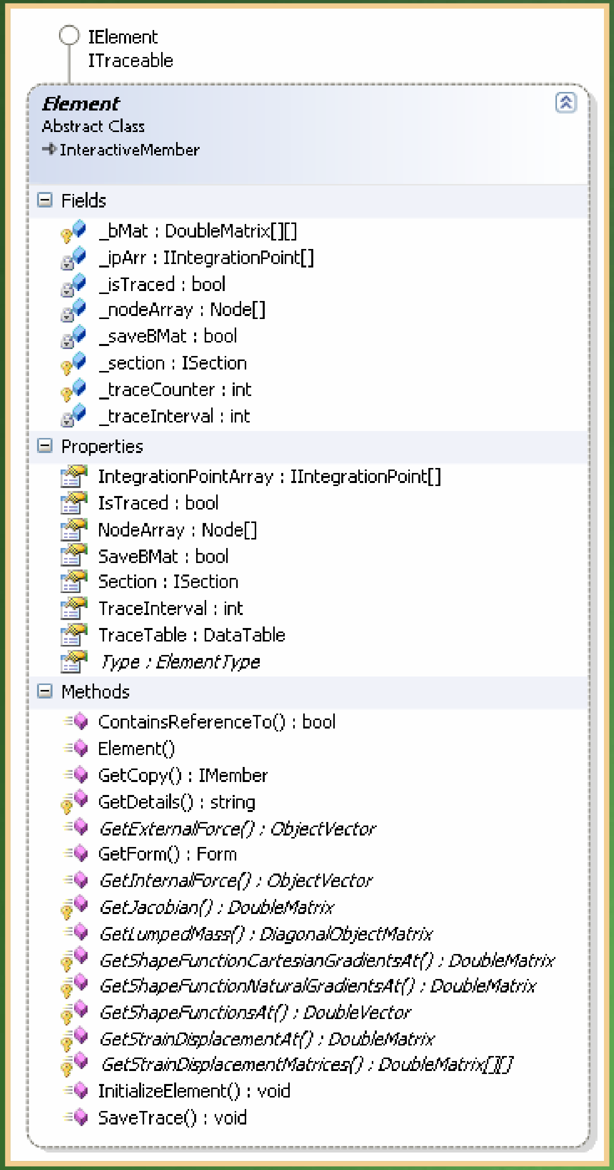
SUBROUTINE BAND(A, B, IMAX, NBW, N)
Solves a banded matrix and returns the displacements.

Outline

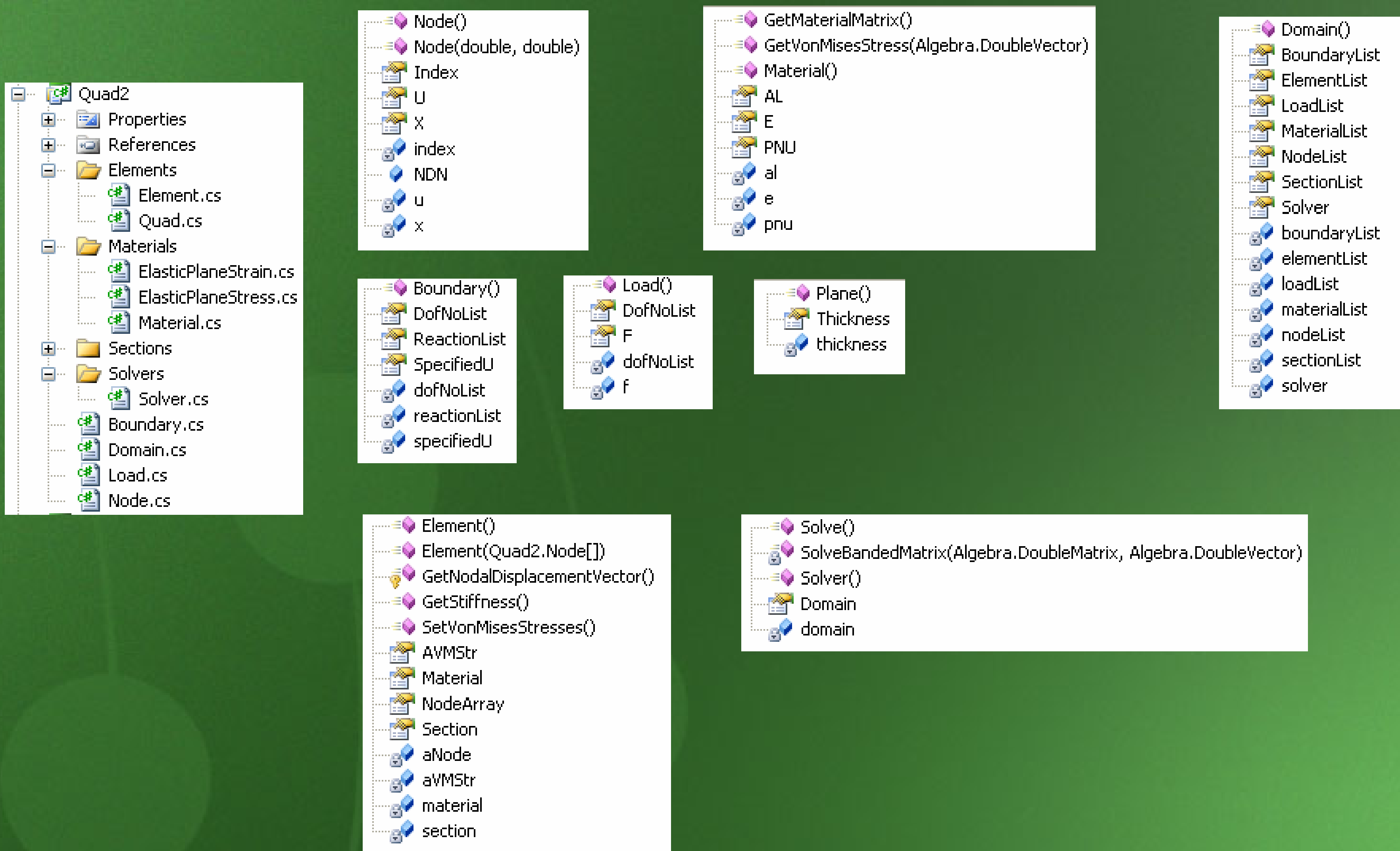
- The .NET Framework
  - The .NET Framework
  - C# Programming Language
- Procedural FEM
  - Data structures
  - Subroutines
- Object-Oriented FEM
  - Classes
- Discussion

Object Oriented FEM

- Classes are the building blocks
- Objects are "instances" of classes
- Objects are combinations of code and data
- Data crunches itself



Classes



Quad2
Properties
References
Elements
Element.cs
Quad.cs
Materials
ElasticPlaneStrain.cs
ElasticPlaneStress.cs
Material.cs
Sections
Solvers
Solver.cs
Boundary.cs
Domain.cs
Load.cs
Node.cs
Node()
Node(double, double)
Index
U
X
index
NDN
u
x
GetMaterialMatrix()
GetVonMisesStress(Algebra.DoubleVector)
Material()
AL
E
PNU
al
e
pnu
Domain()
BoundaryList
ElementList
LoadList
MaterialList
NodeList
SectionList
Solver
boundaryList
elementList
loadList
materialList
nodeList
sectionList
solver
Boundary()
DofNoList
ReactionList
SpecifiedU
dofNoList
reactionList
specifiedU
Load()
DofNoList
F
dofNoList
f
Plane()
Thickness
thickness
Element()
Element(Quad2.Node[])
GetNodalDisplacementVector()
GetStiffness()
SetVonMisesStresses()
AVMStr
Material
NodeArray
Section
aNode
aVMStr
material
section
Solve()
SolveBandedMatrix(Algebra.DoubleMatrix, Algebra.DoubleVector)
Solver()
Domain
domain

Classes

Element

- Element()
- Element(Quad2.Node[])
- GetNodalDisplacementVector()
- GetStiffness()
- SetVonMisesStresses()
- AVMStr
- Material
- NodeArray
- Section
- aNode
- aVMStr
- material
- section

Quad

- GetA(Algebra.DoubleMatrix, double)
- GetG(Algebra.DoubleVector)
- GetIntegrationPoints()
- GetJacobian(Algebra.DoubleVector)
- GetStiffness()
- Quad()
- Quad(Quad2.Node[])
- SetVonMisesStresses()
- Section
- section

Material

- GetMaterialMatrix()
- GetVonMisesStress(Algebra.DoubleVector)
- Material()
- AL
- E
- PNU
- al
- e
- pnu

Plane Stress

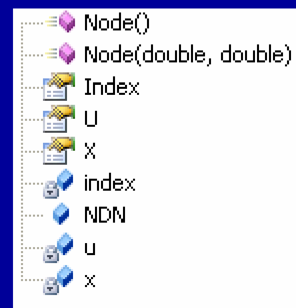
- ElasticPlaneStress()
- GetMaterialMatrix()
- GetVonMisesStress(Algebra.DoubleVector)

Plane Strain

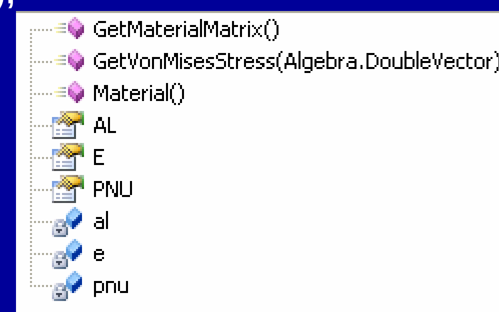
- ElasticPlaneStrain()
- GetMaterialMatrix()
- GetVonMisesStress(Algebra.DoubleVector)

Example

```
public static void RunQuad2Test()
    {
        // Create the domain
        Domain domain1 = new Domain();

        // Create and add nodes to the domain
        Node n1 = new Node(0, 0);
        Node n2 = new Node(0, 15);
        Node n3 = new Node(0, 30);
        Node n4 = new Node(30, 0);
        Node n5 = new Node(30, 15);
        Node n6 = new Node(30, 30);
        Node n7 = new Node(60, 0);
        Node n8 = new Node(60, 15);
        Node n9 = new Node(60, 30);
        domain1.NodeList.AddRange(new Node[] { n1, n2, n3, n4, n5, n6, n7, n8, n9 });
```



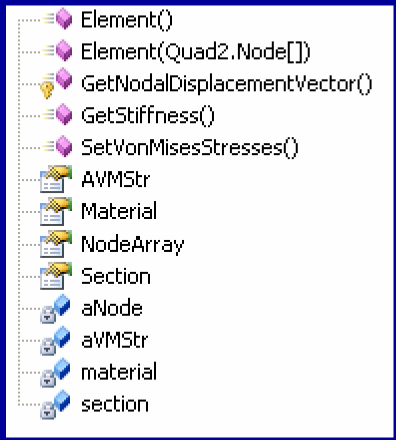

```
        // Create the material and add it to the domain
        Material m1 = new ElasticPlaneStress();
        m1.E = 70000;
        m1.PNU = 0.33;
        m1.AL = 12e-6;
        domain1.MaterialList.Add(m1);
```

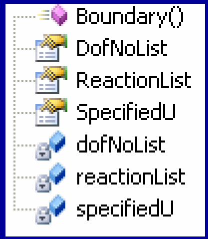


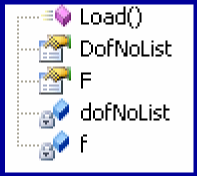

```
        // Create the section and add it to the domain
        Section s1 = new Plane();
        ((Plane)s1).Thickness = 10;
        domain1.SectionList.Add(s1);
```

Example

```
 // Create the elements and add them to the domain.
  Element e1 = new Quad(new Node[] { n1, n4, n5, n2 });
  Element e2 = new Quad(new Node[] { n2, n5, n6, n3 });
  Element e3 = new Quad(new Node[] { n4, n7, n8, n5 });
  Element e4 = new Quad(new Node[] { n5, n8, n9, n6 });
  domain1.ElementList.AddRange(new Element[] { e1, e2, e3, e4 });
  foreach (Element e in domain1.ElementList)
  {
     e.Material = m1;
     e.Section = s1;
  }

  // Create the boundary and add it to the domain
  Boundary b1 = new Boundary();
  b1.SpecifiedU = 0;
  b1.DofNoList.AddRange(new int[] { 0, 1, 2, 3, 4, 5 });
  domain1.BoundaryList.Add(b1);

  // Assign the load
  Load l1 = new Load();
  l1.F = -10000;
  l1.DofNoList.Add(17);
  domain1.LoadList.Add(l1);

  // Start solution
  domain1.Solver.Solve();
}
```







Discussion

- Maintainability
- Extensibility
- Speed