WCCM V
Fifth World Congress on
Computational Mechanics
July 7–12, 2002, Vienna, Austria
Eds.: H.A. Mang, F.G. Rammerstorfer,
J. Eberhardsteiner

An Object Oriented Finite Element Toolkit

Rachid Touzani*

Laboratoire de Mathématiques Appliquées, UMR CNRS 6620
Université Blaise Pascal (Clermont–Ferrand II)
63177 Aubière, France
e-mail: Rachid.Touzani@math.univ-bpclermont.fr

Key words: Object oriented programming, C++, Finite element method

Abstract
We present an object oriented finite element library written in C++. We outline the main motivations in developing such a library. Through a simple example program we show a typical use of the library. We describe the main class categories and typical problems to solve using the library.

1 Introduction

Object oriented programming has become a powerful tool in scientific computing. In fact, the development of performant compilers and template programming techniques [2, 11] enable getting competitive software with respect to Fortran programs. The advantages of object oriented programming are well known. In particular, encapsulation and reusability of the classes provide a good environment for developing reliable and evolutive software. This is particularly appreciated in research centers (academic and industrial) where software maintenance is a big challenge.

We present, in this paper, a library of C++ classes, called **`OFELI`** (as Object Finite Element Library). This library is to be viewed as a toolkit to build up one's own finite element dedicated codes. This approach is, in our opinion, more efficient than the one consisting in working with multipurpose large size finite element codes. These black boxes are in general difficult to understand, adapt or modify in function of specific applications. Of course, developing specialized finite element programs with the help of existing libraries is possible in Fortran but object oriented programming seems to be more adapted to this task (see for this [3, 4] for instance). The unique difficulty resides here in the fact that building a finite element code with the help of **`OFELI`** requires for a developer to be moderately experienced with finite element implementation and C++ programming. The amount of experience depending of the complexity of the problem to solve.

The library is intended to be used for various purposes :

1. *Research* : enable a research scientist (senior or PhD student) to quickly develop and test a method with sufficient flexibility in order to modify a finite element, a solution technique, a time integration scheme, ...

2. *Teaching* : make possible, through well adapted exercises, for a student to run an existing finite element code on an application and make himself a significant contribution by developing a new element, a new equation or a new solution method.

3. *Engineering and industrial applications* : A particular care is taken to preserve performance and efficiency. Moreover, solving coupled phenomena codes is made easily with the help of problem dependent classes.

In the following, we first describe some features of the library through a simple example, then outline the main classes and emphasize on problem dependent classes. These ones being more susceptible to be modified or rewritten by users.

2 An example

Let us illustrate a typical use of the library through a simple elliptic boundary value problem. Consider the problem

$$\begin{aligned} \Delta u &= 0 \qquad \text{in } \Omega, \\ u &= g \qquad \text{on } \Gamma, \end{aligned}$$

where Ω is a domain in $\mathbb{R}^2$ with boundary Γ and g is a given data defined on Γ.

Assume that a finite element mesh of the domain Ω is constructed and stored in file **mesh.m** with a defined format that we will not precise here. A typical C++ program, that uses **OFELI**, has to perform the following steps :

1. Read mesh data and determine matrix structure,
2. Calculate and assemble element matrix and right-hand side,
3. Take account for boundary conditions,
4. Solve the linear system of equations,
5. Output solution.

These steps can be realized through the the following C++ program :

```
#include "OFELI.h"
#include "Therm.h"
using namespace OFELI;
int main()
{
   Mesh ms("mesh.m");
   SkSMatrix<double> a(ms);
   Vect<double> b(ms.NbDOF()), bc(ms.BC());
   Element *el;
   for (ms.TopElement(); (el=ms.GetElement());) {
     DC2DT3 eq(el);
     eq.Diffusion();
     a.Assembly(el,eq.A());
   }
   a.Prescribe(ms,b,bc);
   a.Factor();
   a.Solve(b);
   cout << b;
   return 0;
}
```

From this simple program we can extract the main used tools in **OFELI** to solve a boundary value problem.

– The finite element mesh is stored as an instance of a class (**Mesh**) that is constructed simply by giving the name of the mesh file. Any available data in the mesh can be retrieved by invoking this class methods.

– We use here a class to store the problem matrix. The template class **SkSMatrix<double>** deals with skyline symmetric matrices. Here the template parameter stands for the data type in the matrix. Note that the instance is constructed with the help of mesh information that assigns matrix structure.

– Vectors are to be used with the help of class **Vect<double>**.

– Boundary conditions (here the function g) are read in this simple example in the mesh file and are thus retrieved in the **Mesh** instance. More advanced procedures are available in the library.

– All loops over elements can be performed using methods of class **Mesh**. Here, member function **GetElement()** returns a pointer to class **Element** that contains element information and increments the pointer to next element.

– The only problem–dependent class here is **DC2DT3** (that stands for *Diffusion-Convection, 2–D, 3–Node triangle*). Each term of the equation is added via a class member.

– Finite element assembly (here for the matrix only) is made using a method of the matrix class. The finite element matrix being given from element equation instance.

– The function **Prescribe** allows prescribing Dirichlet (essential) boundary conditions by a penalty method. Other methods are possible.

– Finally, member functions **Factor** and **Solve** factorize the matrix and solve the resulting linear system of equations.

– Output of the solution (and also of matrix, mesh, vectors, ...) can be obtained by using the output stream operator.

3 Classes in OFELI

From the above example we can sort **OFELI** classes into some principal families. In particular, it is clear, that here above, the only problem dependent class is **DC2DT3** (Diffusion–Convection, 2-D, 3–Node triangles), all other classes being common to all problems and constitute a kernel of the library.

3.1 Mesh classes

This family concerns all mesh related classes like **Node**, **Element**, **Side** or **Mesh** with obvious meanings. Here, a finite element mesh is an instance of class **Mesh** that can be constructed by simply giving a mesh file name. A mesh is a collection of instances of classes **Node**, **Element** or **Side**.

3.2 Vector classes

The library contains a wide variety of vector classes to define and manipulate vectors. All these classes are template classes, the template parameter being data type. A basic class called **Vect<T>** derives from the familiar standard **vector<T>** and adds some methods and operators that are specific to scientific computing and finite element techniques. Other vector classes are more finite element oriented.

3.3 Matrix classes

This set of classes provides tools for storing matrices in the mainly used storage schemes in finite element computations. Beside a dense matrix class, there is a skyline matrix class in symmetric **SkSMatrix** and nonsymmetric (**SkMatrix**) versions. Furthermore, the class (**SpMatrix**) deals with sparse matrices (storing only nonzero entries) in order to use iterative solvers.

3.4 Boundary conditions, Forces, ...

A finite element computation needs several types of data (boundary conditions, body forces, initial conditions, . . .). These data can be either given from input file as vectors or via a class written by the developer and that inherits from an abstract class called **`UserData`**.

3.5 Material data

In order to define material properties, a class for each material type is available. The procedure is the following : If a finite element lies in a material (*e.g.* Copper), then the element code (an integer number) for this element has to be associated to the material Copper. From a programming point of view **`Copper`** is a class, and the properties of copper can be accessed via the constructors of this class. Several constructors of material classes are implemented in order to initialize data in several situations (constant data, temperature dependent data, ...). The used materials are to be defined in the mesh file. Note that a default material called **`GenericMaterial`** is defined with default values for coefficients.

3.6 Finite Element Equations

A basic step in the implementation of the finite element method consists in building up finite element equations for each element and assembling them into a global linear system of equations. Of course, a nonlinear problem is solved by an iteration algorithm where each iteration consists in solving a linear problem. The same technique is valid for transient problems where iterations are performed in order to advance in time. Here at the element level, finite element equations are to be declared as instances of a class of finite element equations. Clearly, the finite element equation depends on the problem to solve.

The **`OFELI`** library contains a collection of problem dependent classes that implements models coming from heat transfer, structural mechanics, fluid mechanics and electromagnetics.

3.7 Finite Element Shapes

The above classes need information about finite element interpolation (choice of element geometry, shape functions and their partial derivatives). The **`OFELI`** library provides a family of classes that implement most popular finite element shapes. For instance, class **`Triang3`** corresponds to P_1 (or three-node) triangle.

3.8 Problem solvers

The **`OFELI`** package contains a variety of problem solvers. The term “Problem Solver” means here any class or function that performs any type of analysis.

- Steady state linear problems are solved by using either direct solvers, that are methods of matrix classes, or iterative solvers. These ones are implemented as template functions that enable, in particular using standard preconditioners.
- Nonlinear problems are solved by iterative procedures that are implemented via classes.
- Optimization problems are solved by invoking specific classes. The objective function (and its gradient) is given here by providing a user-defined class that is typically problem dependent.

- Time stepping algorithms can be “manually” realized in the current version. Their usage is explained in examples.

4 A finite element equation class

Let us show how can an equation be implemented as a class in **OFELI**. For this, let us consider the case of linearized elasticity with plane strains using the 4–node quadrilateral. The class is called **Elas2DQ4**. The most important class methods are :

- **Elas2DQ4(const Element *el)**
  This constructor gets element geometric information and initialize element material properties.
- **Mass()**
  This member function adds element mass contribution to element matrix.
- **Deviator()**
  Adds element deviatoric contribution to element matrix.
- **Dilatation()**
  Adds element dilatational contribution to element matrix.
- **double *A()**
  Returns a pointer to element matrix.
- **double *b()**
  Returns a pointer to element right-hand side.

Other methods concerning, for instance, the right-hand side are available but are not presented here. Note that no parameters are to be passed to the equations. Material dependent properties (here Lamé coefficients) are retrieved from element material data. In a simple case, we have a so-called *generic material*, whereas in more realistic cases, the material name is read from mesh file and a ...

5 Availability

The **OFELI** library is available from the web page [10]. It is free but copyrighted. It is distributed under the terms of the GNU General Public License (GPL). The package contains the source files, a large set of examples with increasing complexity and an extensive documentation.

References

[1] E. Arge, A. M. Bruaset, P. B. Calvin, J. F. Canney, H. P. Lantangen, *On the numerical efficiency of C++ in scientific computing*, Numerical Methods and Software Tools in Industrial Mathematics, M. Daehlen and A. Tveito, eds., Birkhäuser (1999), pp. 93–109.

[2] M. H. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*, Addison-Wesley (1999).

[3] P. R. B. Devloo, *PZ: An object oriented environment for scientific programming*, Comp. Meth. Appl. Mech. Eng., 150(1–4): (1997), 133–153.

[4] D. Eyheramendy, Th. Zimmermann, *Object–oriented finite elements I. Theory and application of automatic programming*, Comp. Meth. Appl. Mech. Eng., 1954, (1998), 41–68.

[5] I. Masters, A. S. Usmani, J. T. Cross, R. W. Lewis, *Finite element analysis of solidification using object-oriented and parallel techniques*, Int. J. Numer. Meth. Eng., 40, (1997), 2891–2909.

[6] R. I. Mackie, *Object–oriented finite element programming – The importance of data modelling*, Advances in Engineering Software, 32(9–11): (1999), 775–782.

[7] B. Meyer, *Object–Oriented Software Construction*, 2nd Ed., Prentice-Hill, (1997).

[8] B. Patzak, Z. Bittnar, *Design of object finite element code*, Advances in Engineering Software, 32(10–11): (2001), 759–767.

[9] W. H. Press, S. A. Teukolsky, W. T. Wetterling, B. P. Flannery, *Numerical Recipes in C++, The Art of Scientific Computing, Second Edition*, Cambridge University Press (2002).

[10] R. Touzani, *OFELI, An Object Finite Element Library*, Web Site : **`http://wwwlma.univ-bpclermont.fr/~touzani/ofeli.html`**.

[11] T. Veldhuizen, *Using C++ template metaprograms*, C++ Report, 7, No. 4, (1995), 36–43.