

**OBJECT-ORIENTED FINITE ELEMENT SOFTWARE USING C++****Nacer E. EL KADRI E**

* Department of Mathematics Physics and Informatics, Laboratory - LSI, Polydisciplinary Faculty of Taza, University Sidi Mohamed ben Abdellah, Taza, Morocco

DOI: 10.5281/zenodo.1204213**KEYWORDS:** Object Oriented Programming; Finite Elements; C++; Continuum Mechanics.**ABSTRACT**

The objective of this work is to present the design of finite element software using an object-oriented language. Based on developments in software engineering, object-oriented programming brings new concepts that make it possible to modify the conventional programming approach and to generate software with a flexible and decentralized architecture. Firstly, some defects of the conventional programming approach, such as: illegibility of the software, high cost of maintenance, difficult scalability, etc., will be mentioned. Object-oriented programming will then be proposed as a remedy. We will define the concept, before detailing the main features. Subsequently, the development of finite element software in object-oriented programming will be discussed. A definition and description of each object used will be given. Finally, the programming of this software in C++ language is presented.

INTRODUCTION

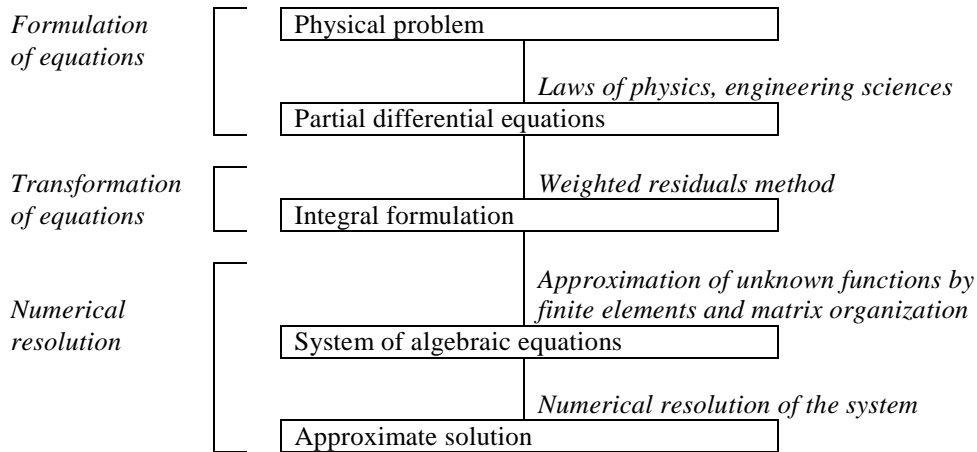
The finite element method (abbreviation FEM) has become one of the most powerful numerical methods ever developed. The FEM has been successfully applied to all branches of continuum mechanics. Among the basic features of the FEM which led to its success, we can mention. The possibility of choosing a variety of variational formulations associated with the mathematical model. The ability to work on unstructured meshes, which facilitates the modeling of complex geometries. Good accuracy at reasonable cost. The natural treatment of boundary conditions. Software programming to handle a wide variety of physical problems governed by partial differential equations.

Effective programming of the FEM requires a good experience both in the field of finite elements and computer science. Finite element software is usually complicated, a few thousand lines of instructions. These programs must perform a wide variety of operations: database manipulation, scientific calculations, post-processing, and so on. Programming must be very efficient to minimize the cost of design. This efficiency can be related to the type of computer used, as well as the computer language. The most advanced finite element software is written in a classic programming language: Fortran, C, etc. In practice, it has been found that the adaptation, maintenance or reuse of these software presents difficulties, particularly in terms of the cost of design or maintenance. Precisely, these types of difficulties come from the usual programming languages used. For a decade, a new philosophy of finite element programming has emerged. This is object-oriented programming (abbreviation OOP). Taking advantage of the development of software engineering, the OOP brings new concepts that make it possible to modify the conventional programming approach and to generate software with a flexible and decentralized architecture, easier to maintain.

Thus, this work presents a contribution to the finite element software design methods in OOP. At first, some defects of the conventional programming approach will be stated: illegibility, high cost of maintenance, difficult scalability, etc. The OOP will then be proposed as a remedy. We will define the concept, before detailing the main features. Subsequently, we will present the development of finite element software in OOP. A definition and description of each object used will be given.

FINITE ELEMENT FORMULATION

To dominate these projects, the engineer needs models that allow him to simulate the behaviour of complex physical systems. The laws of physics (or other laws) can describe the behaviour of these systems through partial differential equations. The FEM is one of the methods used today to effectively solve these equations. The FEM [1-12] consists of using a simple approximation of unknown variables, to transform the system of partial differential equations into a system of algebraic equations, and thus to provide an approximate solution of the problem.



Consider the problem of convection-diffusion, whose physical behaviour is represented by the system of partial differential equations

$$V_{,t} + F_{ii}^{conv}(V) = F_{ii}^{diff}(V) + F \tag{1}$$

in a domain Ω of boundary Γ . With V is the vector of unknown variables, F_i^{conv} is the convection flow in the direction i , F_i^{diff} is the diffusion flux in this same direction and F is the source vector. To this system is added the boundary and initial conditions.

In order to numerically determine a solution, we cancel the projection of the system of equations (1) on a set of weighting functions W

$$\int_{\Omega} \{ W \cdot [V_{,t} + F_{ii}^{conv}(V) - F_{ii}^{diff}(V) - F] \} d\Omega = 0 \tag{2}$$

The second order terms in the system (1) are integrated by parts. The contour integral can be used to impose natural boundary conditions. We thus establish the weak variational form

$$\int_{\Omega} \{ W \cdot [V_{,t} + F_{ii}^{conv}(V) - F] + W_{,i} \cdot [F_i^{diff}(V)] \} d\Omega = \int_{\Gamma} \{ W \cdot [F_i^{diff}(V) \cdot n_i] \} d\Gamma \tag{3}$$

The introduction of the weighting functions and the interpolation functions, in the elementary weak variational form, leads to an elementary matrix system for the nodal values of the approximation. The global form is constructed by assembling elementary forms. This assembly is then organized in matrix form

$$[M]\{V_{,t}\} + [K(V)]\{V\} = \{F\} \tag{4}$$

where $[M]$ is the global mass matrix, $\{V\}$ is the global vector of the nodal variables the global stiffness matrix $[K(V)]$ and $\{F\}$ is the global source vector.



Any program based on FEM includes the following steps:

<i>Reading, checking and organizing data describing a mesh</i>	
Read and print: - The coordinates of the nodes - Boundary Conditions - Physical properties - The connectivity of the elements	
<i>Construction of global matrices and global vectors</i>	
Construction of global matrices and global vectors For each element: - Extract information related to this element - Construct elementary matrices and elementary vectors, [m], [k] and [f] - Construct global matrices and global vectors. assemble [m], [k] and [f] in [M], [K] and [F]	
<i>Resolution of the system of algebraic equations</i>	
$[M]\{V_n\} + [K(V)]\{V\} = \{F\}$	
<i>Printing results</i>	

FINITE ELEMENT METHOD AND OBJECT ORIENTED PROGRAMMING

The OOP consists firstly of identifying, in the form of classes, the main actors of the FEM. Each actor must be able to perform certain tasks by interacting with other objects. The choice of classes is certainly the most important step when designing software in OOP. Most of the researchers [12-21] who have done work on the application of OOP at the FEM considers a super class to build matrices and elementary vectors. Based on the inheritance principle, they define classes whose function is to gather all the data necessary for the super class. The assembly and the resolution are carried out, either with the help of another class or by enrichment of the already existing classes.

Our goal is to design a software with a simple and flexible architecture, able to deal with various problems by the FEM (solids mechanics, fluid mechanics, thermal, etc.). The work of the researchers [13-21] and our experience in finite elements [1, 12] led us to identify the following classes of objects:

- The *TMesh* class deals with the organization of the data of a mesh;

<i>TMesh class</i>
<i>Member data:</i> number of nodes total number of elements total number of dimensions of the problem number of degrees of total freedom number of equations total number of boundary conditions total ...
<i>Methods :</i> read in a file create a TElement element ...



- The *TNode* class deals with the information related to the nodes of the mesh;

<p><i>TNode class</i></p> <p><i>Member data:</i></p> <ul style="list-style-type: none"> node number number of dimensions number of degrees of freedom vector coordinates vector numbers of equations vector degrees of freedom ... <p><i>Methods:</i></p> <ul style="list-style-type: none"> give your spatial dimension give your coordinates give your equation numbers give your degrees of freedom ...

- The *TPrel* class deals with nodal and elementary physical properties;

<p><i>TPrel class</i></p> <p><i>Member data:</i></p> <ul style="list-style-type: none"> number of groups of physical properties number of physical properties vector of physical properties ... <p><i>Methods :</i></p> <ul style="list-style-type: none"> gives the physical property of the node gives the physical property of the element ...

- The *TElement* class deals with the information related to the elements of the mesh;

<p><i>TElement class</i></p> <p><i>Member data:</i></p> <ul style="list-style-type: none"> number of the element number of nodes per element element type ... <p><i>Methods :</i></p> <ul style="list-style-type: none"> gathers all the information related to the element gathers information related to the physical properties of the element add to the list of TMesh objects ...

- The *TCDE* class deals with the resolution of systems of partial differential equations.

<p><i>TCDE class</i></p> <p><i>Member data:</i></p> <ul style="list-style-type: none"> number of time steps the time step ... <p><i>Methods:</i></p>
--



read the initial solution
 calculate the residual vector
 solve the discrete system by the method of Newton-Raphson
 solve the discrete system by the GMRES method
 ...

Our choice of class is distinguished by the fact that the data structure relative to the mesh (classes *TElement*, *TNode*, *TPrel* and *TMesh*) is managed according to a hierarchy, whereas, the operations characteristic with the method of the finite elements (class *TCDE*) are managed according to another hierarchy. In this way, the hierarchy of the data structure can be reused for other purposes, such as: designing a software for building and / or adapting mesh, etc.

***TMesh* class description**

The first function of the *TMesh* class is to be the interlocutor with the user, that is to say, to read the data files of a mesh (node coordinates, boundary conditions, physical properties and connectivity of elements), which other classes do not have access to. This measurement is simple because only *TMesh* is able to manage all data files and know the entire mesh. *TMesh* must read data in a certain order: node coordinates boundary conditions, physical properties, and connectivity of elements. The reason is that the *TElement* class uses pointers to nodes and physical properties. These pointers are created when reading connectivity.

The second function of the *TMesh* class is to receive this data in order to allocate memory space and create each new object of the FEM: node, physical property, element, etc. Each object is placed, subsequently, in the corresponding list and will thus be accessible by any other object.

***TNode* class description**

Knowing that one of the entities of the element is the node. Which leads us to the creation of the *TNode* class. The node as an object encapsulates all the essential information attached to it. These can only be accessed by a message sent to the object. Thus, the node as object represents a decentralized environment.

***TPrel* class description**

An approach similar to creating the *TNode* class, leads to the *TPrel* class. In a finite element problem, one can have two types of physical properties: nodal and / or elemental. Each type of physical property can contain one or more property groups. These groups will be associated with some or all of the nodes or elements of the mesh. The physical property, as an object, presents a characteristic example of the decentralization of information: the search for physical properties is carried out by the *TPrel* class and not by the element itself. As it should, the method "gives the physical property ..." does not depend on the element.

***TElement* class description**

The *TElement* class will take advantage of the objects of the element's entities intensively by fitting into an already existing environment. In this environment of objects, the element is thus identified by its number and must above all use: its nodes, its physical properties, etc. The role of the *TElement* class is to group all the information (objects) related to an element of the mesh. This information forms the element object that is subsequently added to the object structure of the *TMesh* class.

The interest of the encapsulation of the information appears clearly: the search for the physical properties is carried out by the entity physical property, that is to say the class *TPrel*, and not by the element itself. In the same way, the search for the coordinates of the nodes of the element is carried out by the *TNode* class and not by the element; as well as many other data (member data of the class in question) that will no longer depend directly on element. The software can therefore really be written in decentralized, extensible and reusable modules.

***TCDE* class description**

The *TCDE* class was designed as part of the numerical resolution of partial differential equations. This application consists more particularly in determining the approximate solution of these equations, at each time step. It is then



Global Journal of Engineering Science and Research Management

necessary to obtain the discretization of these equations by the finite element method in space and by finite differences in time.

To obtain the spatial discretization, it is necessary to acquire the information and to carry out the following computations for each element of the mesh:

1. Information about the element.
2. The type of elements (linear, triangular, tetrahedral, prismatic, etc.).
3. The type of weighting functions.
4. The type of numerical integration method used, in order to know the number of integration points and to calculate the numerical weight.
5. The construction of the mass matrix, the stiffness matrix, the nodal solution vector, and the source vector.

For temporal discretization: Euler schema, retrograde differentiation, etc. The following solutions must be formed for each element of the mesh:

6. The solution at time t , and/or at times $t + \Delta t$ and $t - \Delta t$.

To obtain the overall solution, it is necessary to assemble in order to obtain:

7. The global matrix system.
8. The global residue vector.

For numerical resolution, it is necessary:

9. The numerical method (Gauss, Newton-Raphson, GMRES, etc.).

The following development will enrich the objects defined in the previous study on the mesh and also define the class *TCDE*.

The *TMesh* class translates step 1 of spatial discretization. The *TMesh* object perfectly meets this need. Thus, an item identified by its number encapsulates all the necessary and available information to calculate the contributions that are described in steps 2 to 5.

In order to calculate the contributions and thus go through steps 6 to 8, the *TMeshCFD* class uses a *TElementTCDE* element that was created to gather elementary data for the *TCDE* class more quickly. The classes *TMeshCFD* and *TElementTCDE* respectively derive classes *TMesh* and *TElement*.

Note that an interesting option, easy and simple implantation, can be used thanks to OOP. This option consists in enriching the *TMesh* class, that is to say to make it organize all the data necessary for spatial discretization. This amounts to saying that an element identified by its number is an object that encapsulates the information from step 1 to 5. However, this option has not been retained, in order to preserve the database relating to the mesh in a hierarchy and the characteristic operations to the finite element method in another hierarchy. Through specialized classes (*TMeshCFD* and *TElementTCDE*), the interface between the two hierarchies is realized.

The classes *TMeshCFD* and *TElementTCDE* group all the data and methods common to any finite element. Similarly, the *TCDE* class will regroup the set of data and methods common to any time integration scheme (Euler, retrograde differentiation, etc.) defining the problem in finite element space. *TCDE* as object has the approximate solution of the problem at each time step.

**APPLICATION EXAMPLE - COMPUTATIONAL OF FLUID DYNAMICS (CFD)**

This software has been used successfully for the simulation of a viscous compressible flow around the NACA0012 profile on a mesh of 8 150 elements and 16 496 nodes, for a number of equations to be solved of 40 636. The mesh is shown in Fig. 1a, with a zoom around the profile on Fig. 1b.

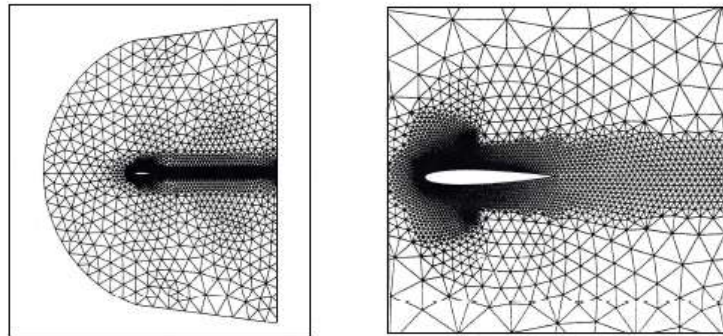


Figure 1: a) Mesh around the profile NACA0012. b) Zoom around the profile NACA0012

Mach contours, density contours and velocity fields are illustrated in Figs. 2a, 2b and 2c respectively.

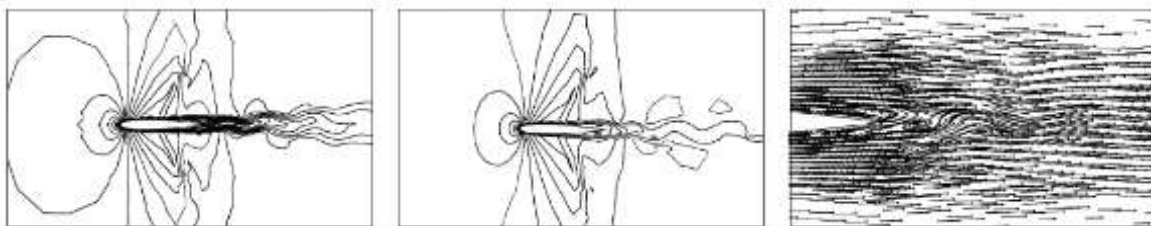


Figure 2: a) Mach contours. b) Density contours. c) Velocity

We note a gain of about 22.4% on runtime compared to equivalent Fortran software [1, 12]. On the other hand, we note a gain of 8.9% compared to the software in Fortran, as for the time necessary only to manipulate the data (organization and transfer of the data relative to the mesh), apart from all arithmetic operations, always in the case of flow around the profile NACA0012.

CONCLUSION

A finite element software, having a simple and flexible architecture, has been programmed in C++ language. This software has been used successfully for solving the Navier-Stokes and Euler equations. Note that the description of the software presented reflects its current state which is not definitive, since the developments still continue in order to realize software in OOP for the resolution of partial differential equations and the construction and / or adaptation of mesh. Compared to software designed in Fortran language, the object-oriented approach has brought a big change in the organization and design of this new software. This new way of programming improves maintenance, robustness, scalability and re-usability. The characteristics of OOP that led to this result are the encapsulation of data, the hierarchical organization of the software associated with the concept of inheritance, polymorphism and sending message.

REFERENCES

1. N.E. Elkadri E., A. Soulaïmani, C. Deschênes, "A finite element formulation of compressible flows using various sets of independent variables", *Computer Methods in Applied Mechanics and Engineering*, vol. 181, issues 1–3, pp. 161-189, January 2000.
2. B. Guo, I. Babuska, "The h-p version of the finite element method," *Computational Mechanics*, vol. 1, issue 1, pp. 21–41, March 1986.



Global Journal of Engineering Science and Research Management

3. M. Ainsworth, J.T. Oden, "A posteriori error estimation in finite element analysis," Wiley, New York, 2011.
4. J.M. Melenk, B.I. Wohlmuth, "On residual-based a posteriori error estimation in hp-FEM," *Advances in Computational Mathematics*, vol. 15, Issue 1–4, pp. 311–331, November 2001.
5. J.C. Nedelec, "Mixed finite elements in \mathbb{R}^3 ," *Numerische Mathematik*, vol. 35, issue 3, pp 315–341, September 1980.
6. F. Brezzi, M. Fortin, "Mixed and hybrid finite element methods," Springer, Berlin, 1991.
7. D.N. Arnold, R.S. Falk, R. Winther, "Finite element exterior calculus, homological techniques, and applications", *Acta Numerica*, vol. 15, pp. 1-155, May 2006.
8. M. Neilan, D. Sap, "Stokes elements on cubic meshes yielding divergence-free approximations," *Calcolo*, vol. 53, issue 3, pp 263–283, September 2016.
9. T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, issues 39–41, pp. 4135-4195, October 2005.
10. B. Cockburn, J. Gopalakrishnan, R. Lazarov, "Unified Hybridization of Discontinuous Galerkin, Mixed, and Continuous Galerkin Methods for Second Order Elliptic Problems," *SIAM, Journal on Numerical Analysis*, vol. 47(2), pp. 1319–1365, February 2009.
11. L. Demkowicz, J. Gopalakrishnan, "A class of discontinuous Petrov–Galerkin methods. Part I: The transport equation," *Computer Methods in Applied Mechanics and Engineering*, vol. 199, issues 23–24, pp. 1558-1572, April 2010.
12. Nacer E. Elkadri E., "Une méthode d'éléments finis pour la dynamique des gaz et conception orientée objet du code de calcul," Ph.D., Thèse de doctorat, Département de génie mécanique, Université Laval, Québec, Canada, 1996.
13. W. Bangerth, R. Hartmann, G. Kanschat, "deal.II—A general-purpose object-oriented finite element library," *ACM Transactions on Mathematical Software (TOMS)*, vol. 33, issue 4, August 2007.
14. W. Bangerth, D. Davydov, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, D. Wells, "The deal.II Library, Version 8.4," *Journal of Numerical Mathematics*, vol. 24, issue 3, pp. 135–141, 2016.
15. M.S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, "The FEniCS Project Version 1.5," *Archive of Numerical Software*, vol. 3, no. 100, pp. 9–23, 2015.
16. P.T. Bauman, R.H. Stogner, "GRINS: A Multiphysics Framework Based on the libMesh Finite Element Library," *SIAM Journal on Scientific Computing*, vol. 38(5), pp. S78–S100, October 2016.
17. C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, S.J. Sherwin, "Nektar++: an open-source spectral/element framework," *Computer Physics Communications*, vol. 192, pp. 205-219, July 2015.
18. MOOSE (Multiphysics Object-Oriented Simulation Environment) Framework. <http://mooseframework.org/>
19. MFEM—a free, lightweight, scalable C++ library for finite element methods. <http://mfem.org/>
20. F. Hecht, "New development in freefem++," *Journal of Numerical Mathematics*, vol. 20, No. 3-4, pp. 251-265, 2012.
21. A. Dedner, M. Nolte, "Construction of Local Finite Element Spaces Using the Generic Reference Elements," *Advances in DUNE*. Springer, Berlin, Heidelberg, pp. 3-16, 2012.