COMPUTATIONAL MECHANICS
WCCM VI in conjunction with APCOM'04, *Sept. 5-10, 2004, Beijing, China*
© 2004 Tsinghua University Press & Springer-Verlag

FER/View – An Interactive Finite Element Post-Processor

Z.Q. Feng[*], Z.G. Feng

Laboratoire de Mécanique d'Evry, Université d'Evry, 40, rue du Pelvoux, 91020 Evry, France
e-mail: feng@iup.univ-evry.fr

Abstract Development of user-friendly and flexible scientific programs is a key to their usage, extension and maintenance. This paper describes the main features of FER/View, an interactive finite element post-processor developed by using an OOP (Object-Oriented Programming) approach and the graphic standard OpenGL. This software has been specifically designed to fulfill most of the common requirements of engineers and numericians, in the context of finite element simulations. This program is rather intuitive with a friendly GUI and, therefore, the user does not really need any specific learning stage prior to be able to use it. Several post-processing examples with graphical representations illustrate some functionalities of the program: to visualize and manipulate mesh data structures via the mouse, to deal with scalar/tensor values associated with a mesh and to deal with more advanced features (e.g., interactive animations, cutting planes or texture transparency, etc.).

Key words: post-processing, object-oriented programming, graphical user interface, finite element

INTRODUCTION

Numerical modeling is a powerful technique for the solution of complex engineering problems. The finite element method is the most used in engineering computations. One of the significant requirements in the design of a scientific computing program is the ability to store, retrieve, and process data that maybe complex and varied. Numerous applications of numerical simulations based on finite element/volume methods involve a mesh of the computational domain as spatial support. In this context, it is often necessary to visualize this mesh and the solutions associated with mesh entities. This is typically the aim of a scientific visualization tool (also called a post-processing tool). Hence, the use of graphical devices allows, provided the software is well-suited to the problem considered, the user to appreciate and even to analyze a mesh and/or a solution. To be efficient and pertinent, this tool must be fast and should offer various features, if possible, in a user-friendly environment (*i.e.*, convivial and interactive). To the users of such a program, it is important not only to have a powerful solver, but also to work in a convivial graphical interface environment. On the other hand, as the problems to solve have grown in size and complexity, the codes have also grown with complex mathematical procedures and data control. This places a high demand for maintenance, new developments and re-use on the programming strategy and language chosen. It exists a lot of finite element analysis programs such as ANSYS, ABAQUS, etc. which have their own post processors. There are also some specific post-processing tools such as GiD [1], Tecplot [2] …

The aim of this paper is to present the development of the program FER/View: an interactive finite element post processor with friendly graphical user interfaces. FER/View cannot be considered as a substitute to the existing graphical tools, but it is more a simple alternative to users that need to quickly look at a result within a convivial interface rather than a complex GUI.

The primary goal of the present paper is to illustrate the capabilities and effectiveness of FER/View. Section 1 outlines some concepts of the object-oriented approach in developing scientific programs. Section 2 describes the general organization of FER/View and its main features. Section 3 gives several post-processing examples to illustrate some functionalities of FER/View, described in Section 2.

1. Object-Oriented Programming in C++

This section describes some concepts and terminology of object-oriented programming. The basic concept of object-oriented programming is the encapsulation of data structure and a set of functions (procedures) manipulating the data in prepackaged software components called *objects*. By using an object-oriented language such as C++, a natural way of manipulating finite element objects such as node, element, boundary conditions, matrix, vector can be adopted. The notion of "object" has been widely employed in many computer science fields. As compared with the traditional function-oriented programming technique, object-oriented programming is more structured and modular, yielding programs that are easily maintained, resilient, and powerful because of its basic features: data abstraction, encapsulation and data-hiding, modularity, classes, hierarchy and inheritance, polymorphism and dynamic binding etc. However, this notion is not largely used in the field of numerical simulation. We know that most programs in scientific computing are written in FORTRAN, in which it is difficult to write structural and object-oriented programs even though a concerted effort has been made in this field. Of the many possible programming languages, C++ is being increasingly used in engineering applications because it was designed to support data abstraction structure and object-oriented programming. In addition, C++ is the extension of the popular language C. So it is becoming the first choice for scientists and engineers to develop object-oriented programs for the analysis of engineering problems [3-8].

The benefit of an object-oriented approach in C++ is mainly due to the definition of classes. A class is defined by the groups of objects that have the same kind of data and procedures. Objects are called instances of a class in the same sense as standard FORTRAN variables are instances of a given data type. The class concept can be reviewed as an extension of the *record* concept in Pascal or *struct* concept in C to provide a set of attached procedures acting on the data. Unlike conventional programming techniques, which require the developer to represent data and procedures separately, an object in C++ is a user-defined and self-contained entity composed of data (private or public) and procedures. This allows developers to design objects which know how to behave. Box 1 shows an example of the class ELEMENT used in FER/View.

```
class ELEMENT  {
protected:
            int m_number;               // element number
            int m_nnel;                 // number of nodes
            int m_imat;                 // material number
            VECTINT elem_node;          // element connectivity
 public:
            static NODE *node;
            static CONTROL *control;
            static POSTPRO *postpro;
            static BOOL CalculateNormal(float*, float*, float*);
            virtual void DrawMesh(){};
            virtual void DrawContour(){};
            virtual void WriteAsc(fstream& ASC);
            int getNumero(){ return m_ number; };
            ELEMENT();
            ELEMENT(int id, int *elemtype, int *elemnode);
            virtual ~ELEMENT();
};
```

Box 1. ELEMENT class definition

In this class, m_number, m_nnel, etc. are protected data of type integer, and elem_node is an instance of the class VECTINT that is another user-defined class defining an integer vector. DrawMesh, WriteAsc, etc. are member functions (procedures) of object ELEMENT. ELEMENT() and ~ELEMENT() are respectively the default constructor and destructor. A high-level object can be created by assembling a group of objects. This new object has the collective functionality of its sub-objects. This concept of object abstraction can be extended to the complete software applications. A program assembled from objects can itself be an object and thus be included in another program. This method has been applied when building the solid mechanics analysis program FER/Solid by adding the finite element solver into FER/View [9].

One of the fundamental techniques in object-oriented programming is the use of *inheritance*. Inheritance is a way of creating new classes called derived classes, which extend the facilities of existing classes by including new data and function members, as well as changing existing functions. For instance, the development of FER/View requires the creation of class CFerMechApp which is the derived class of MFC class CWinApp for Windows applications [10] (Box 2).

```
class CFerMechApp : public CWinApp {
public:
        CFerMechApp();
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CFerMechApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL
        //{{AFX_MSG(CFerMechApp)
        afx_msg void OnAppAbout();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

Box 2. Derived classes

It is noted that objects communicate with each other by sending messages. When an object receives a message, it interprets that message, and executes one of its procedures. That procedure operates on the private data of the object. So the internal details of how it functions are hidden from the program that uses the object. This also means that the internal function of an object could be modified without having to change the rest of the program. Thus, the program becomes modular and easy to read. Without entering into the details, Fig. 1 shows the principal database mapping of FER/View.

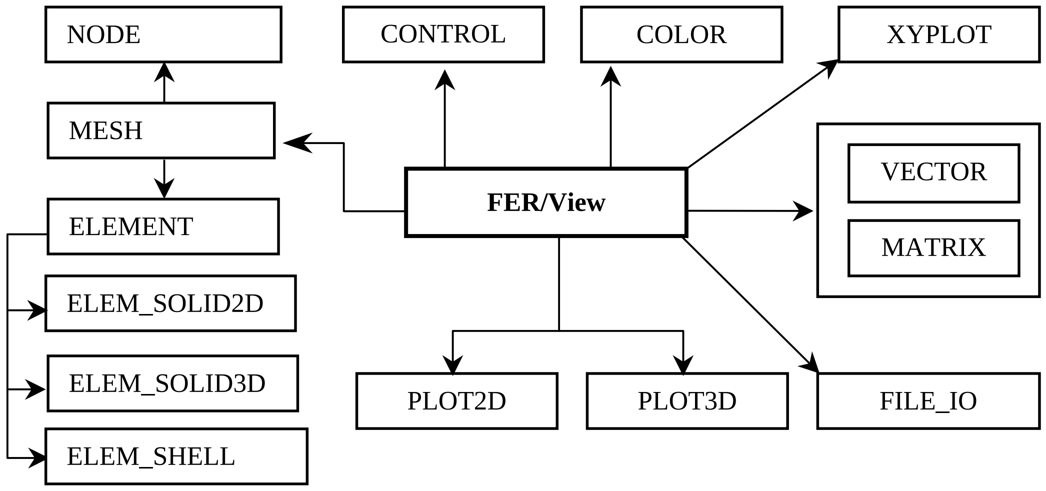


Fig. 1. Principal class diagram of FER/View

The class CONTROL stores some control flags about light, plot symmetry, animation, dynamic rotation or move, etc. XYPLOT stores the numerical result date and procedures for time history curve plot of dynamic response. ELEM_SOLID2D, ELEM_SHELL, etc. are inherited from the base class ELEMENT.

It is worth noting that many components of FER/View have directly been used to create another multi-body dynamics analysis program FER/Mech [10]. Object-oriented programming makes this possible and allows rapid development of new software. *Reusability* is so becoming a key issue in software development.

2. General Organization of FER/View

FER/View is an integrated environment composed of several functional modules. Fig. 2 summarizes the main ones.

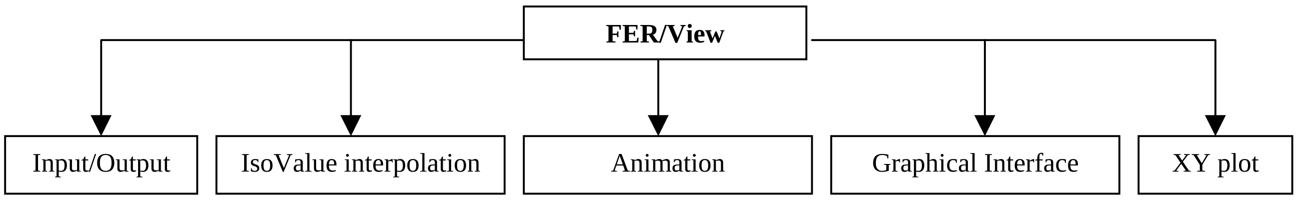
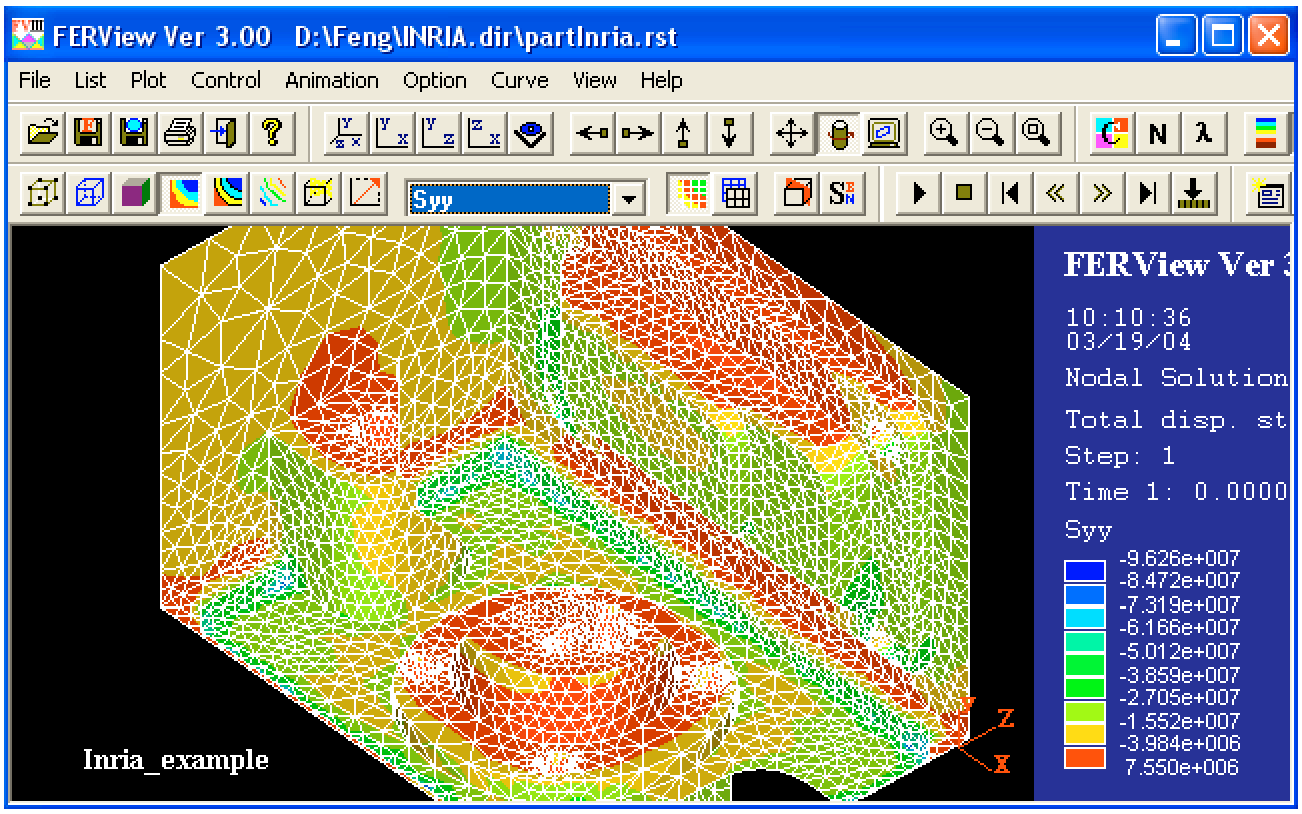


Fig. 2. General organization of FER/View

MFC (Microsoft Foundation Class) [11] has been proposed by Microsoft for the easy development of Windows applications. In this project, MFC is largely used to design the user-interface objects such as Dialog boxes, Menu, Icons, Tool Bar, String Table, etc. OpenGL [12] is a relatively new industry standard that in only a few years has gained an enormous following. It is now a standard graphics library integrated in Windows or UNIX systems. OpenGL is a procedural rather than a descriptive graphics language. Instead of describing the scene and how it should appear, the programmer actually describes the steps necessary to achieve a certain appearance or effect. These steps involve calls to a highly portable API (Application Programming Interface) that includes approximately 120 commands and functions. These are used to draw graphics primitives such as points, lines, and polygons in three dimensions. In addition, OpenGL supports lighting and shading, texture mapping, animation, and other special effects. A lot of these capabilities have been implemented in FER/View. The result is satisfactory. The user-interface of the program is shown in Fig. 3.



```
FERView Ver 3
10:10:36
03/19/04
Nodal Solution
Total disp. st
Step: 1
Time 1: 0.0000
Syy
```

🟦	-9.626e+007
🟦	-8.472e+007
🟦	-7.319e+007
🟩	-6.166e+007
🟩	-5.012e+007
🟩	-3.859e+007
🟩	-2.705e+007
🟩	-1.552e+007
🟨	-3.984e+006
🟥	7.550e+006

Fig. 3. Friendly user-interface of FER/View

FER/View has many functionalities, some main features being summarized as follows:

- Load the model (neutral file) and the results (output file)
- Available not only for the general finite element static or dynamic analysis, but also for the model with variable numbers of nodes and elements during computation (i.e. using remeshing method)
- Plot node, element mesh and geometry shape
- Display mesh deformation. All visualization techniques use the deformed mesh with undeformed mesh overlay
- Display different colors according to the element property
- Display node, element or material number
- Plot contours, iso-line, iso-surface and vector
- Display minimum and/or maximum values of data for the current time step or for all time steps
- Change the color division number for contour or iso-line etc. plot
- Add or cancel light effects and wire frame
- Display selected elements group
- Display time history of multiple data
- Display the distribution curve of data on a line path
- Use mouse operation for rotation, pan and zoom as well as node and element picking
- Save and print images and curves
- Animate the results in any case of display mode
- Plot cutting plane
- Construct automatically feature edges
- Show transparent view.

3. Demonstrated examples

In order to illustrate and validate the functions of FER/View, discussed above, several post-processing examples of different nature have been realized. It is interesting to note that although the program is of small size (1.07 Mo), it can be used for different engineering problems: solid mechanics, fluid dynamics, geometry analysis, etc.

3.1 Plasma spray process: a multi-physics problem

We consider a fully molten pure aluminum particle impacting onto a rigid steel substrate with simultaneous heat transfer [13]. Fig.4 shows the deformation of the mesh during the impact process. It is noted that the mesh is always uniform in the computation due to an automatic remeshing technique. The number of elements is so changed at each time step. Such situation makes it difficult to use the commercial software such as ANSYS, I-DEAS, etc. to visualize the results. This particular case is accounted for in FER/View. In addition, a three-dimensional animation can be generated from the bi-dimensional computations and the velocity field can be represented by the vector plot (Fig.5).

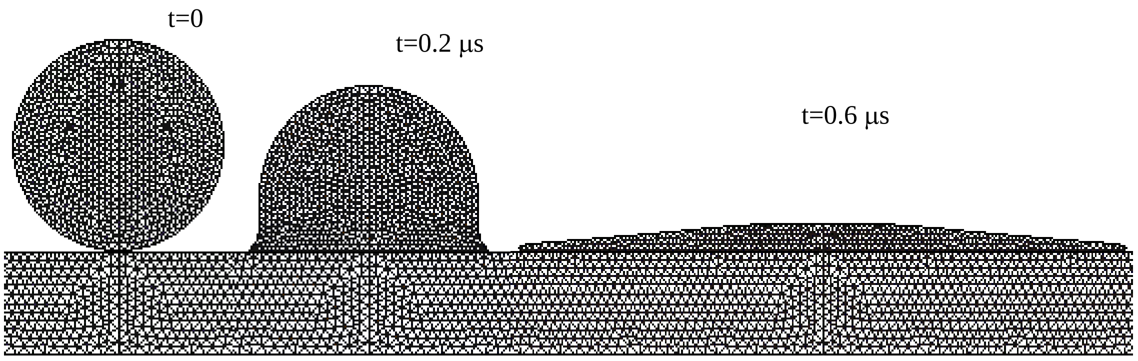


Fig.4. Plasma spray process: deformed meshes at different time

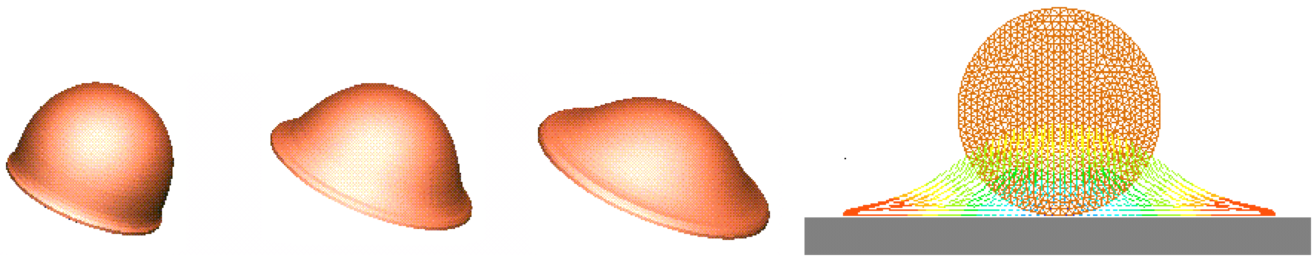


Fig.5. Three-dimensional animation and vector plot

3.2 Performance test

In order to show the efficiency of FER/View, we propose two tests performed on a PC with the OS Windows XP. The specifications of the PC used are: Pentium4, 2.8Ghz, 1 Go (SDRam at 333 MHz), video Nvidia GeForce Ti4200 128 Mo, hard disc IDE 80 Go.

The first test deals with a mesh with 150 779 tetrahedral elements and with one sequence of results (three translations and seven stress components on each node), as shown in Fig. 3. The elapsed time to open the file and to plot the model is about 3s. The animation, rotation, zoom, pan,… are quite fluid.

The second example is the biggest model viewed up to now by FER/View: 1329865 nodes and 2673604 three-dimensional triangular elements (Fig. 6). The elapsed time to open the file and to plot the model is about 30s.

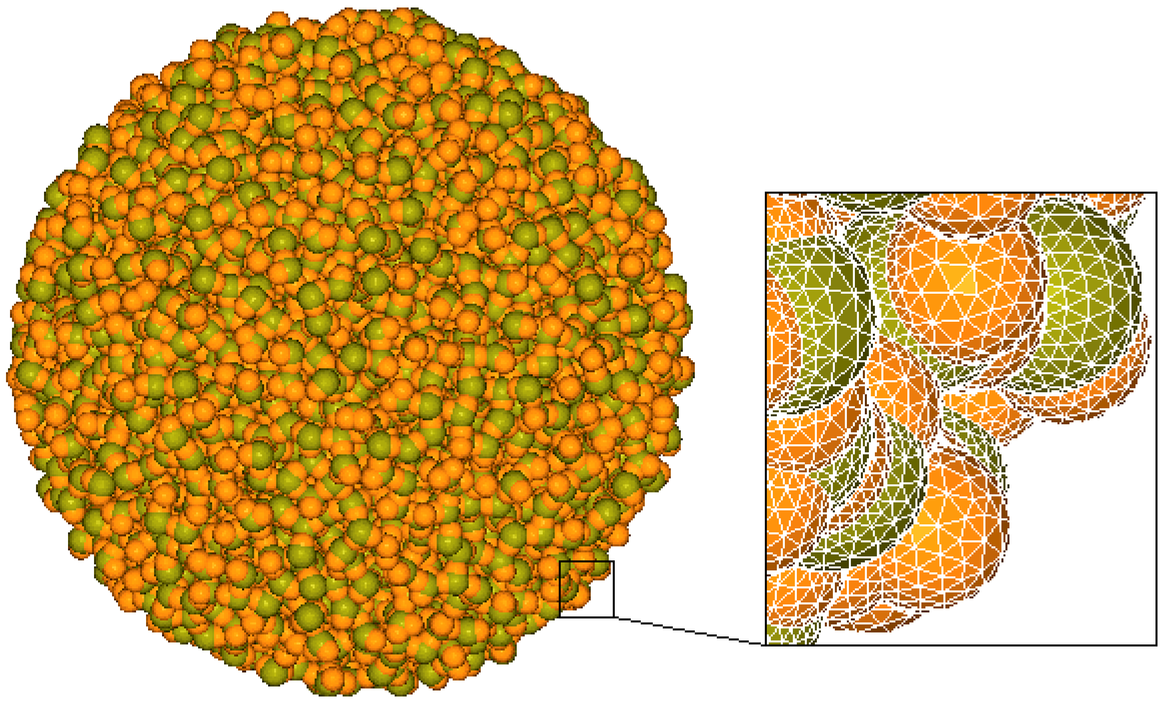


Fig.6. Huge model viewed by FER/View

3.3 Iso-surface and streamline plot in CFD – Computational Fluid Dynamics

The first CFD example simulates the incompressible swirling flow inside a cylindrical tank, generated by the rotation of one lid [14]. This model is composed of 320000 nodes and 313632 three-dimensional hexahedral elements. Fig.7 displays the iso-surfaces of the fluctuating axial velocity with respect to a steady axisymmetric base flow. It is worth noting that the generation and the visualization of the iso-surfaces take only about 2s.

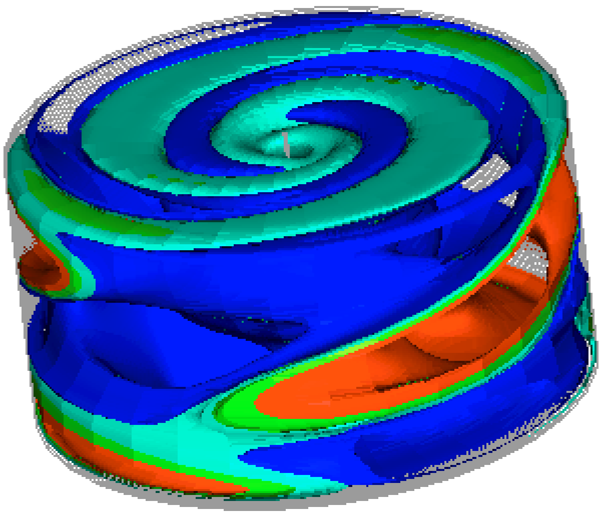


Fig.7. Iso-surface plot

The second CFD example deals with the aerodynamic flowfield around a rectangular obstacle [15]. Fig.8 shows the streamlines from which we can see the separation bubbles and the vortex shedding.

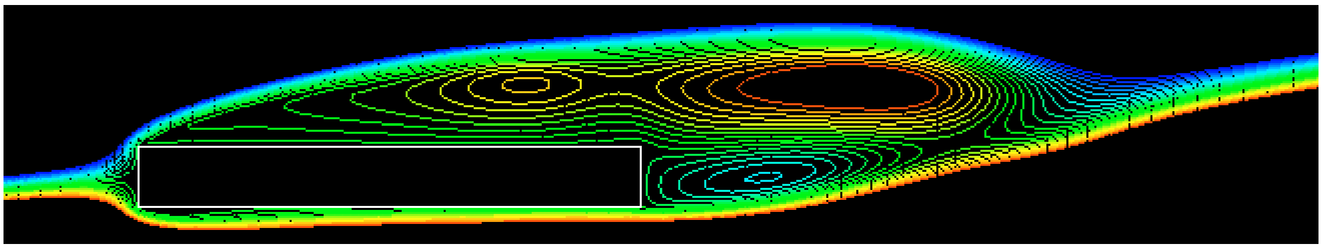


Fig.8. Streamlines plot

3.4 Cutting plane plot

Sometimes, when modeling complex three-dimensional solids, we need to see what happens inside the structure. This task can be easily realized by picking three nodes directly on the screen to define a cutting plane, as shown in Fig. 9.

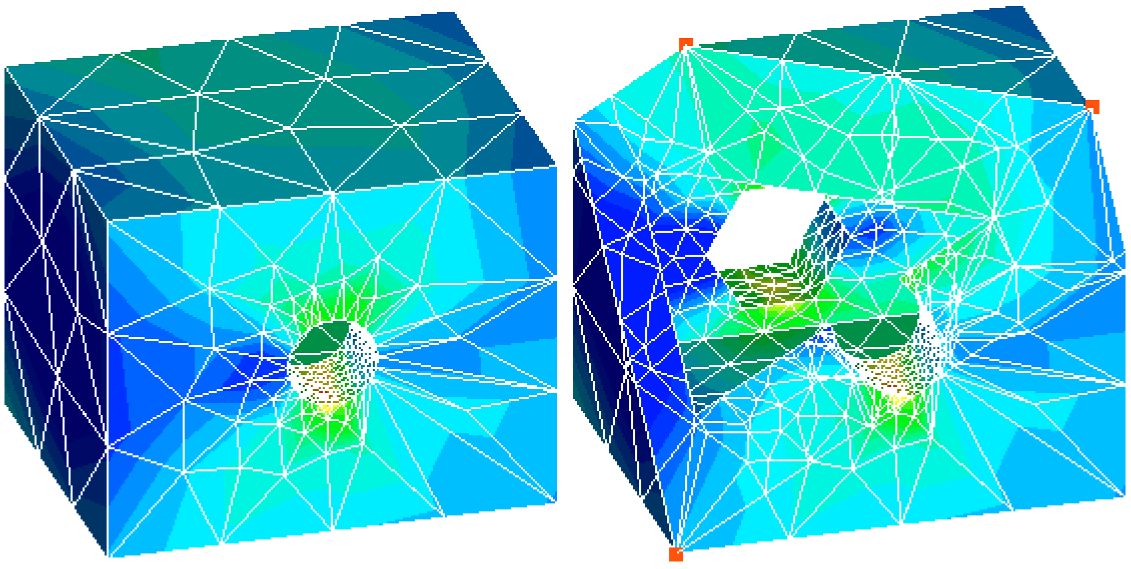


Fig.9. Cutting plane

3.5 Transparent view and feature edge

In FER/View, it is also possible to make some areas to be transparent in order to have a global view of complex structures, as shown in Fig. 10 and Fig. 11. From the mesh model (i.e. the nodes and the elements), we can create automatically the feature edges with FER/View, as shown in Fig. 11.

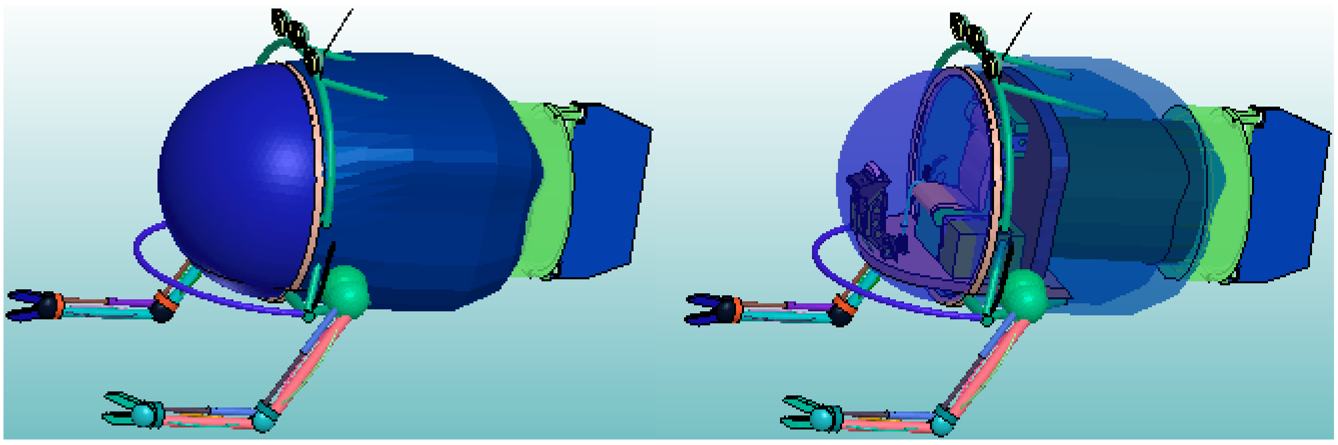


Fig.10. Transparent view

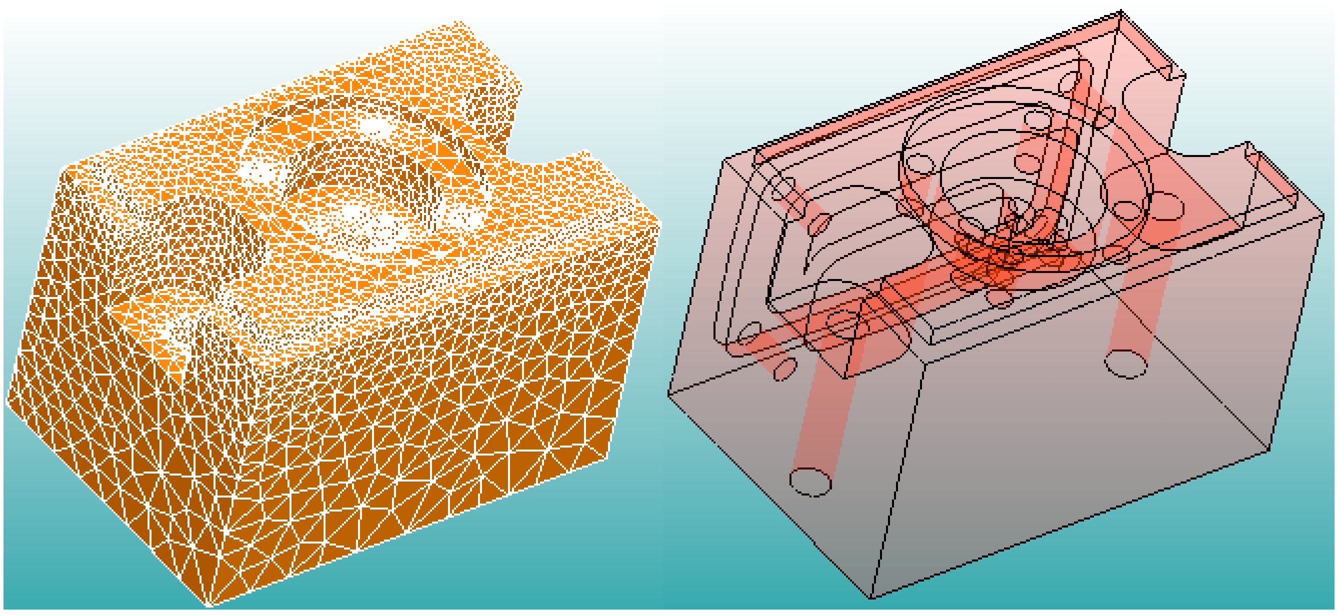


Fig.11. Feature edges

4. Conclusions

In this paper, we have presented the software prototype FER/View. The development of FER/View has been focused on the simplicity, speed, effectiveness and the conviviality. The open architecture of the program due to the object-oriented programming technique facilitates further developments and adapts to suit specific needs easily and quickly. Moreover, the proposed graphical user interface has proved to be satisfactory and flexible as we can see from the post-processing examples. Some advanced features have been developed in FER/View and the results are encouraging as illustrated by the examples. We would conclude that FER/View helps effectively the solver developers to examine their results quickly and easily. The authors feel confident that object-oriented programming in C++ will promote the development of computational tools for engineering sciences and GUI applications.

Acknowledgements The authors gratefully acknowledge Prof. Olivier Daube and Dr. Grégory Turbelin for providing the CFD examples. The mesh models (Fig. 6 and Fig. 10) are downloaded from the site of INRIA [16].

REFERENCES

[1] http://gid.cimne.upc.es/

[2] http://www.tecplot.com/

[3] B.W.R. Forde, R.O. Foschi, S.F. Stiemer, *Object-oriented finite element analysis*, Computers & Structures, 34, (1990), 355-374.

[4] R.I. Mackie, *Object-oriented programming of the finite element method*, Int. J. Num. Meth. Eng., 35, (1992), 425-436.

[5] S.P Scholz, *Elements of an object-oriented FEM++ program in C++*, Computers & Structures, 43, (1992), 517-529.

[6] Z.Q. Feng, K. Aazizou, F. Hourlier, *Modélisation des problèmes de contact avec frottement – Implantation en C++ dans le code ZéBuLoN*, Colloque National en Calcul des Structures, 11-14 mai Giens, France, 2, (1993),1141-1156.

[7] Y. Duboispelerin, T. Zimmermann, *Object-oriented finite element programming, 3. An efficient implementation in C++,* Comput. Meth. App. Mech. Eng., 108, (1993), 165-183.

[8] G.W. Zeglinski, R.P.S. Han, P. Aitchison, *Object-oriented matrix classes for use in a finite element code using C++*, Int. J. Num. Meth. Eng., 37,(1994), 3921-3937.

[9] http://gmfe16.cemif.univ-evry.fr:8080/~feng/FerSystem.html.

[10] Z.Q. Feng, P. Joli, N. Séguy, *FER/Mech - A software with interactive graphics for the dynamic analysis of multibody systems*, Advances in Engineering Software, 35, (2004), 1-8.

[11] M. Brain, L. Lovette, *Developing professional applications for Windows 95 and NT using MFC*, Prentice Hall PTR, 1997.

[12] R.S. Jr. Wright, M. Sweet, *OpenGL Superbible: the complete guide to OpenGL programming for Windows NT and Windows 95*, Waite Group Press, (1996).

[13] Z.G. Feng, Z.Q. Feng, M. Domaszewski, G. Montavon, *Lagrangian method and remeshing technique for modeling of spreading of liquid particle during plasma spray process*, IDMME'98 – 2nd International Conference on Integrated Design and Manufacturing in Mechanical Engineering, Compiègne, France, 1, (27-29 May, 1998), 175-182.

[14] E. Barbosa, O. Daube, *Multiple solutions for three-dimensional swirling flow*, C.A. Mecanique, 330, (2002), 791-796.

[15] G. Turbelin, R.J. Gibert, *CFD calculations of indicial lift responses for bluff bodies*, Wind & Structures, 5, (2002), 245-256.

[16] http://www-rocq1.inria.fr/Eric.Saltel/download/