FER/Mech - A software with interactive graphics for dynamic analysis of multibody system

Zhi-Qiang Feng, Pierre Joli, Nicolas Seguy

FER/Mech — a software with interactive graphics for dynamic analysis of multibody system

Zhi-Qiang Feng[a,*], Pierre Joli[b], Nicolas Seguy[b]

[a]*Laboratoire de Mécanique d'Evry (CEMIF-IME), Universite d'Evry-Val d'Essonne, 40 rue du Pelvoux, 91020 Evry, France*

[b]*Laboratoire de Système Compdu Pelvoux, 91020 Evry, France*

Development of user-friendly and flexible scientific programs is a key to their usage, extension and maintenance. This article presents an Object-Oriented Programming approach for the development of FER/Mech—a software with interactive graphics for use in the design and analysis of two and three dimensional multibody dynamic systems. The general organization of the developed software system is given which includes the solver and the pre/postprocessors with a friendly Graphical User Interfaces. The concept of absolute natural coordinates is discussed to model rigid bodies in order to satisfy the constraints of modularity. Two case studies with graphical representations illustrate some functionalities of the program.

Keywords: Multibody dynamics; Object-oriented programming; Graphical user interface

1. Introduction

Numerical modeling is a powerful technique for the solution of complex engineering problems. 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. 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.

Object-Oriented Programming (OOP) is a well-known topic to computer scientists, but somewhat neglected in the computational engineering community. One reason for this is the limited exposure of engineers to computer science concepts. Another reason (historical reason) is that most scientific computing programs (e.g. finite element analysis programs) have been and are being written in a procedural programming language such as FORTRAN. Because of its design, FORTRAN does not encourage the use of data structures other than the array. The OOP techniques are not supported by the language itself. Consequently, the analysis programs are not easily modified for implementing new ideas and new algorithms. However, since several years, this problem has come to the attention of the engineering profession, and much progress has been made to improve the reliability of methods for finite element analysis and to make it easier for usage, extension and maintenance of analysis programs.

In the eighties, several researchers began work on data management in structural analysis software [1–6]. In 1986, Touzot [7] introduced an interactive conception system SIC. One year after, De Saxcé [8] presented the project CHARLY. Verpeaux et al. [9] presented the CASTEM finite element program. These three programs aimed at providing a veritable language devoted to finite element modeling, based on the object database concept. One of the first detailed applications of the object-oriented paradigm to finite element analysis was published in 1990 by Forde et al. [10]. The authors abstracted out the essential components of the finite element method (elements, nodes, materials, boundary conditions, and loads) into a class structure used

* Corresponding author. Tel.: +33-1-69-47-75-01; fax: +33-1-69-47-75-99.

E-mail address: feng@iup.univ-evry.fr (Z.-Q. Feng).

by most subsequent authors. Also presented was a hierarchy of numerical objects to aid in the analysis. Other authors [11–14] increased the general awareness of the advantages of object-oriented finite element analysis over traditional FORTRAN based approaches. Some researchers have concentrated on the development of numerical objects. Scholz [15] gives many detailed programming examples for full vector and matrix classes. Zeglinski et al. [16] provide a more complete linear algebra library including full, sparse, banded, and triangular matrix types. Also included is a good description of the semantics of operators in C++. Lu et al. [17] present a C++ numerical class library with additional matrix types such as a profile matrix. They report efficiency comparable to a C implementation. Zimmermann et al. [18–20] have developed a software architecture in C++ and in SmallTalk for linear dynamic finite element analysis, with extensions to account for material nonlinearity [21]. A freeware, named FreeFem +, was proposed by Pironneau et al. [22] for solving Partial Differential Equations in two dimensions. To our knowledge, the first large-scale finite element analysis program, named ZÉBULON, entirely rewritten in C++, is presented in 1993 by Aazizou et al. [23,24].

All the developments mentioned above are based on numerical computation on physical behavior of structural components. Recently, several finite element analysis programs including a GUI environment such as FER/View, FER/Solid, FER/Contact, etc. have been developed by Feng et al. and are reported in a WEB site [25]. The aim of this article is to present the development of the program FER/Mech. It is composed of several functional modules: the finite element solver for modeling of multibody dynamics, the pre and postprocessors with friendly GUIs. FER/Mech can be considered as a 3D Computer Aided Design system which enables users to create, modify and manipulate a multibody system intuitively and easily in 3D space. From this point of view, it is close to the concept of Virtual Reality Computer Aided Design developed recently by Gao et al. [26].

The primary goal of the present article is to illustrate the practical application of the object-oriented approach to design the engineering software. Section 2 presents the object-oriented approach in developing finite element program. Section 3 describes the general organization of FER/Mech and its main features. Section 4 gives two case studies to illustrate some functionalities of FER/Mech, described in Section 3.

2. Object-oriented programming in C++

This section introduces some concepts and terminology of OOP. The basic concept of OOP 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, OOP 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 OOP. In addition, C++ is the extension of the popular language C. So it becomes the first choice for scientists and engineers to develop object-oriented programs for the analysis of engineering problems.

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. Fig. 1 shows an example of class which defines the font of FER/Mech graphics interface.

In this class, Arial12Normal, Arial14Normal, etc. are private data of type integer, and m_window is an instance of

```
class OGLFONT {
private:
      int    Arial12Normal,Arial14Normal;
      int    Courier12Normal,Courier12Bold,Courier14Normal;
      int    Arial16Normal,Arial16Bold;
      int    Time20Bold;
      VECTINT m_window ;
public:
      void  Establish_Font(HDC ghDC,PRINCIPAL& PRINCIPAL);
      int    FontCreateBitmaps(HDC hdc,      // Device Context
             char  *typeface,                // Font specification
             int   height,                   // Font height/size in pixels
             int   weight,                   // Weight of font (bold, etc)
             DWORD italic);                  // Text is italic
      void  FontDelete(int font);           // Font to delete
      void  FontPuts(int font,char *s);     // String to display
      OGLFONT(); ~ OGLFONT();
};
```

Fig. 1. OGLFONT class definition.

the class VECTINT that is another user-defined class defining an integer vector. Establish_Font, FontCreateBitmaps, etc. are member functions (procedures) of object OGLFONT. OGLFONT() and ~OGLFONT() 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 adding the finite element solver into FER/Mech.

One of the fundamental techniques in OOP 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/Mech requires the creation of class CFerMechApp which is the derived class of MFC class CWinApp for Windows applications [27] (Fig. 2).

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. 3 shows the principal database mapping of FER/Mech.

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_BEAM, ELEM_SHELL, etc. are inherited from the base class ELEMENT. The class RIGIDBODY includes different geometrical shapes such as sphere, cylinder and brick.

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

Fig. 2. Derived classes.

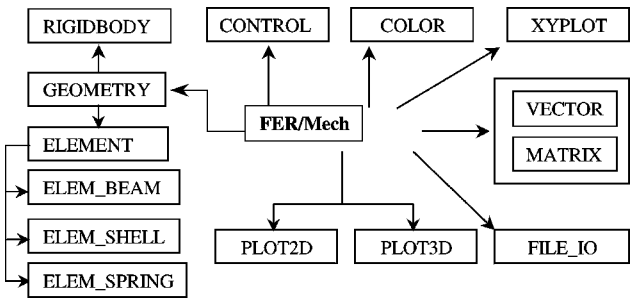

Fig. 3. Principal class diagram of FER/Mech.

It is worth noting that many components of FER/Mech were directly taken from the general purpose finite element postprocessor FER/View developed previously [28]. OOP makes this possible and allows rapid development of new software. *Reusability* is so becoming a key issue in software development.

3. General organization of FER/Mech

FER/Mech is an integrated environment composed of several functional modules. Fig. 4 shows the flow diagram of the software.

After preprocessing, an input file is created, which is used by the solver. The results are written in an output file and displayed by the postprocessor.

3.1. Numerical solution

The solver is based on the finite element approach for solving multibody dynamics problems [29] Currently the code offers static analysis, direct implicit transient analysis and eigenvalue analysis. These analyses lead to the numerical solution of linear or nonlinear systems. Generally, the finite element formulation of the problem of multibody dynamics can be written in the discrete form

$$[M]\{\ddot{u}\} + [C]\{\dot{u}\} = \{F_{\text{int}}\} + \{F_{\text{ext}}\} \tag{1}$$

where the vectors $\{F_{\text{int}}\}$ and $\{F_{\text{ext}}\}$ denote, respectively, the internal, and external forces. $[M]$ is the mass matrix and $[C]$ the damping matrix. $\{\dot{u}\}$ is the velocity vector and $\{\ddot{u}\}$ the acceleration vector. It is noted that the stiffness effect is taken into account by the internal forces vector $\{F_{\text{int}}\}$. The most common method for integrating the dynamics equation

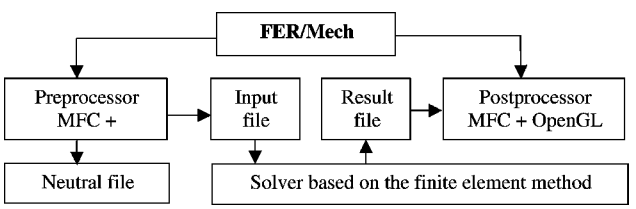

Fig. 4. Flow diagram of FER/Mech.

(1) is the second order Newmark method [30]. It is based on the following assumptions concerning the relation between displacement, velocity and acceleration

$$\{u^{t+\Delta t}\} = \{u^t\} + \Delta t\{\dot{u}^t\} + \Delta t^2\left[(0.5-\alpha)\ddot{u}^t + \alpha\ddot{u}^{t+\Delta t}\right] \quad (2)$$

$$\{\dot{u}^{t+\Delta t}\} = \{\dot{u}^t\} + \Delta t\left[(1-\beta)\ddot{u}^t + \beta\ddot{u}^{t+\Delta t}\right] \quad (3)$$

The parameters α and β determine the stability and precision of the algorithm. With these approximations, the nonlinear dynamics equation. (1) is transformed to an incremental and recursive form for the current iteration $i+1$

$$\begin{aligned}[\hat{K}]^i\{\Delta u\} &= \{F_{\text{ext}}\}^{t+\Delta t} + \{F_{\text{int}}\}^i + \{F_{\text{acc}}\}^i \\ \{u\}^{i+1} &= \{u\}^i + \{\Delta u\}\end{aligned} \quad (4)$$

with the effective stiffness matrix defined by

$$[\hat{K}]^i = [K]^i + \frac{\beta}{\alpha\Delta t}[C]^i + \frac{1}{\alpha\Delta t^2}[M]^i \quad (5)$$

where$[K] = \partial\{F_{\text{int}}\}/\partial\{u\}$ is the tangent stiffness matrix.The inertia forces vector is given by

$$\{F_{\text{acc}}\}^i = -\frac{1}{\alpha\Delta t^2}[M]^i\{u^i - u^t - \Delta t\dot{u}^t - \Delta t^2(0.5-\alpha)\ddot{u}^t\} \quad (6)$$

Recently, a first order time stepping algorithm is applied by Feng et al. [31] to model the impact behavior of deformable bodies.

Because a Multibody System is articulated, we have to consider constraint forces and torques acting onto the joints which can be represented by Lagrange's multipliers. In this case, the equations of motion are solved in conjunction with the constraint equations and it leads to the resolution of Differential Algebraic Equations (DAE). Many methods exist to solve DAE, which can be classified as follows

- Methods to calculate Lagrange's multipliers in two different ways
  - By an explicit formulation such as substitution techniques [32] and penalty functions associated with an explicit or semi-implicit numerical scheme [33]. This can introduce the problem of stabilization of constraints [34]. In addition, in the case of penalty functions, it is difficult to define appropriate values of penalty factors.
  - By an implicit formulation like Hilbert Hughes Taylor (HHT) algorithm [35] which is an adapted Newton Raphson method to avoid the problem of high frequencies of the dynamic response. Another efficient way is based on the acceleration-based augmented Lagrangian formulation using an iterative process to compute $\{\ddot{u}\}$ in order to correct violation of the algebraic constraints at each time [36]. This last numerical technique has the advantage to succeed in the case of singular positions or redundant constraints.
- *Methods to eliminate Lagrange's multipliers*. These techniques consist of partitioning dependent and independent coordinates and consequently, the system of motion/constraint equations is reduced [37]. The problem is the high numerical cost and the ill-conditioned matrix in the case of singular position or redundant constraints. Using the pseudo-inverse constraint matrix by Singular Value Decomposition (SVD) may be an alternative solution [38].

If $\{u\}$ represents only the absolute displacement vector of nodal points, then the mass matrix will be constant. In the case of rigid bodies, large rotations have to be taken into account that change the vector $\{u\}$ into a pseudo-vector because the addition's rule of vectors is no longer valid. In this case, the mass matrix is time dependent. An alternative way is to use natural coordinates, which does not change the nature of the vector $\{u\}$ and moreover leaves the mass-matrix constant [32,36]. However, we have to consider associated algebraic constraint equations (and Lagrange's multipliers) because natural coordinates are not independent.

We do not discuss the very efficient reduced-coordinate (joint coordinate) formulation [39,40] because it does not lead to modular and extensible systems in which connections between bodies can be added or removed during the simulation (contact problems, clearance in the joints …). The modularity is very important to respect the concept of OOP in Finite Element Modeling (FEM). The maximal-natural coordinate formulation creates a number of Degrees Of Freedom (DOF) which is not very important in regard to the number of DOF in a classical FEM. Another advantage of natural coordinates is that the constraint matrix associated to the joints is constant or linear, thus it is possible to use easily the substitution method in an explicit way. Our preference is about a mixed formulation already used to resolve contact problem [41]. This technique is quite similar to the substitution method but its formulation is based on the incremental displacement vector $\{u\}$ which is more suitable to the implicit solution of the system of Eq. (4) presented above.

3.2. Windows utilities and functionalities of FER/Mech

Microsoft Foundation Class (MFC) [27] 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 [42] is a relatively new industry standard that in only a few years has gained an enormous following. It is now

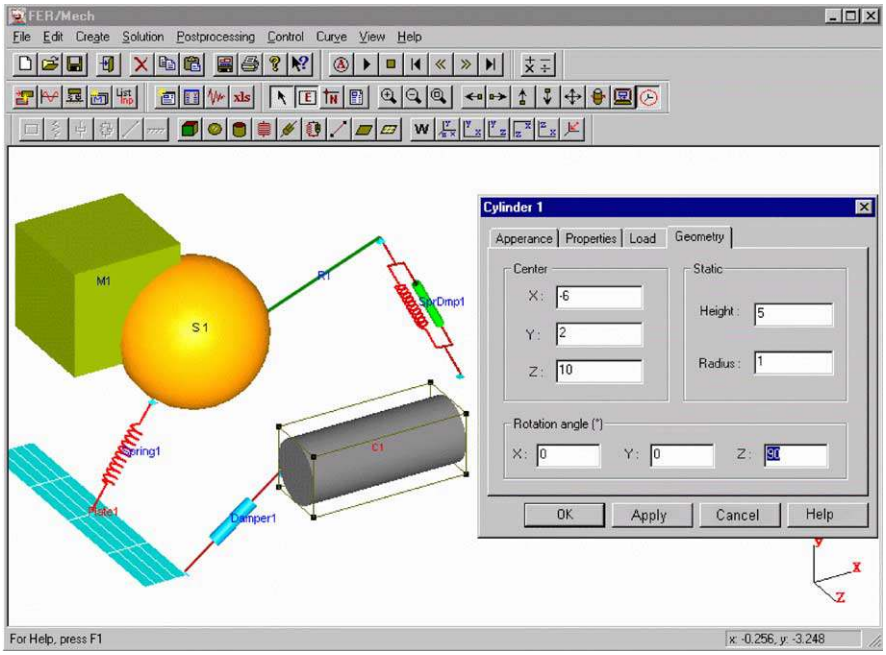

Fig. 5. Friendly user-interface of FER/Mech.

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 Application Programming Interface (API) 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/Mech. The result is satisfactory. The user-interface of the program is shown in Fig. 5, which shows equally some primitive objects and a dialog box.

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

3.2.1. Preprocessor

- Create or delete easily (with icons) different geometrical and mechanical objects like mass, spring, etc.
- Modify the objects (size, orientation and position) and input physical properties (mass, stiffness, initial conditions) with dialog boxes, as shown in Fig. 5.
- Establish the link between the objects to form a mechanism system.
- Apply boundary conditions and loads on the model.
- Input solution control parameters with dialog boxes.
- Display node, element and geometry with or without numbering.
- Select nodes, and elements with dialog boxes and mouse operations.
- List nodes, elements, materials.

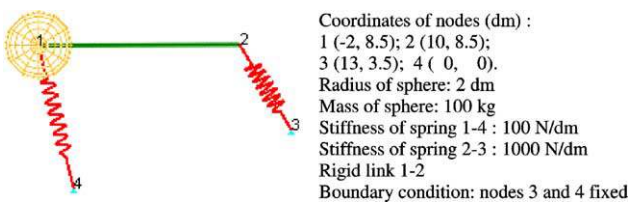

Fig. 6. Free drop of system ‘mass-spring-rigid link’.

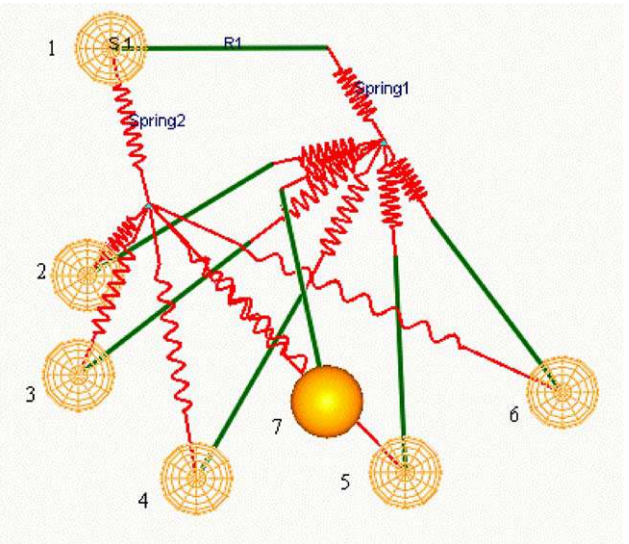

Fig. 7. Animation of deformed shapes.

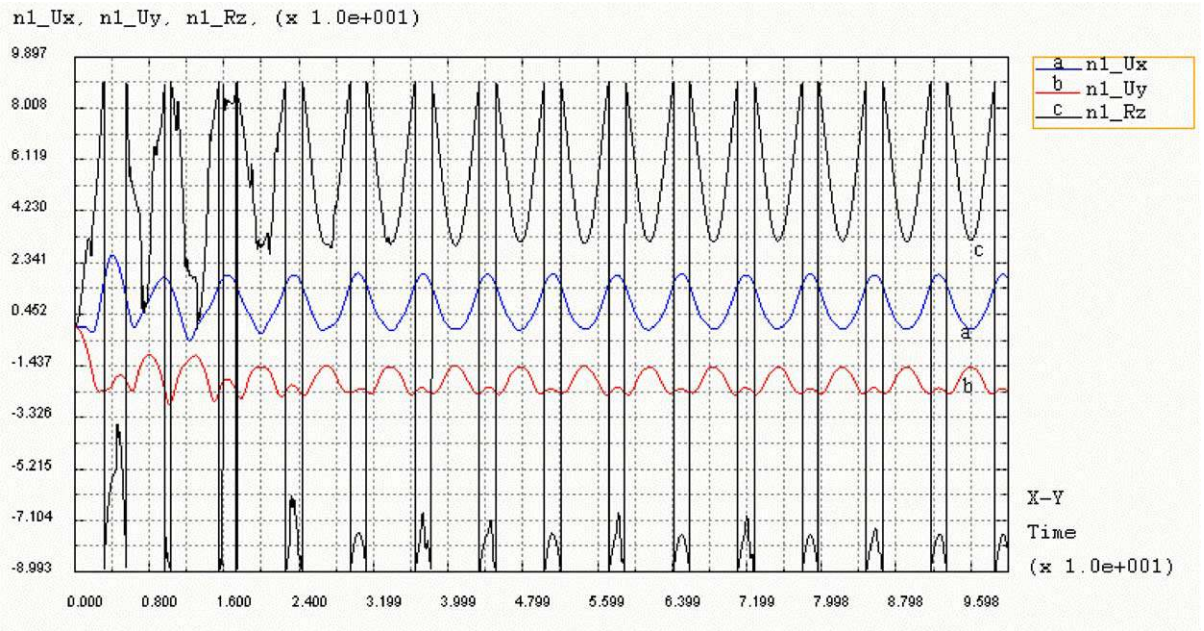

Fig. 8. Displacements (Ux and Uy) and rotation (Rz) of sphere versus time.

- Create a neutral file to save the model. This file contains all information about the model including graphical entities. The graphical entities for the representation of special elements (spring, dampers, mass, etc.) are made of finite elements entities (nodes and elements).
- Create an input file for the solver. This compact ASCII file contains information for the solver with a large place for the comments. This file is easy to read and contains only necessary information (no graphical entities).

3.2.2. Solver

- Include different finite elements such as mass, spring, damper, beam, shell

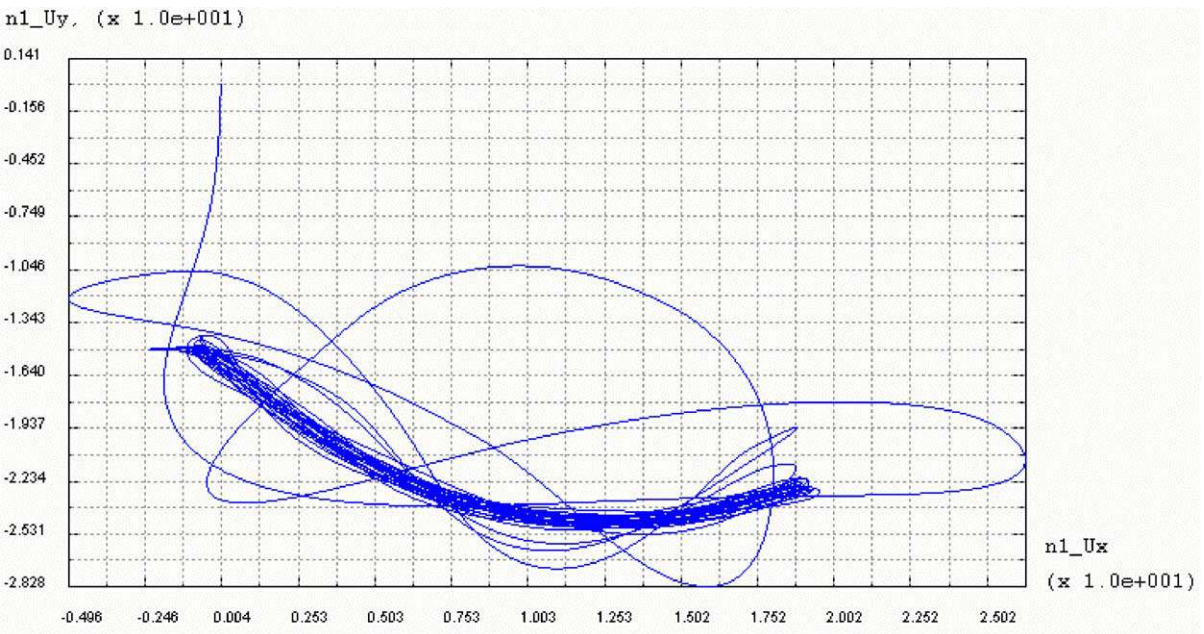

Fig. 9. Trajectory of sphere.

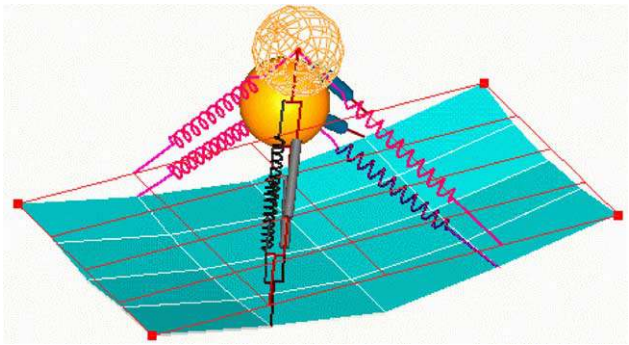

Fig. 10. Rigid sphere coupled to flexible plate.

- Use implicit time stepping schemes like Newmark and Wilson
- Treat geometrical non-linearity due to large displacements
- Perform static and dynamic analyses
- Create an output file for postprocessing.

3.2.3. Postprocessor

- Load the model (neutral file) and the results (output file)
- Display mesh deformation and mode shape of structures
- Add or cancel light effects and wire frame
- Animate the results in any case of display mode
- Display selected elements group
- Display time history of multiple data
- Use mouse operation for rotation, pan and zoom as well as node and element picking.

4. Case studies

In order to test and validate the functions of the GUI, discussed above in the preprocessor and the postprocessor, two examples have been carried out. The possibility to create, to remove or to apply connections between rigid or flexible components opens the imagination of the operator to create various models.

The 'mass-spring-rigid link' example (Fig. 6) tests the robustness of the numerical scheme.

It is 2D constraint system because the rigid link use maximal natural coordinates which are the absolute displacement of each extremity of the link (four DOF). The constraint is that the length of the link has to be constant during the whole time simulation. Moreover, the directions of the two springs change every time that implies a non-constant stiffness matrix. Fig. 7 shows the animation of deformed shapes.

FER/Mech has also the possibility to show the time history of a variable or displacement path, as shown in Figs. 8 and 9.

The 'rigid sphere coupled to flexible plate' example (Fig. 10) tests the possibility to connect a rigid body with structural components modeled by finite elements. But in this case the rotation of the sphere is not taken into account because natural coordinates in 3D are not yet implemented. Fig. 10 shows the initial and deformed positions. The displacements of the sphere are plotted in Fig. 11.

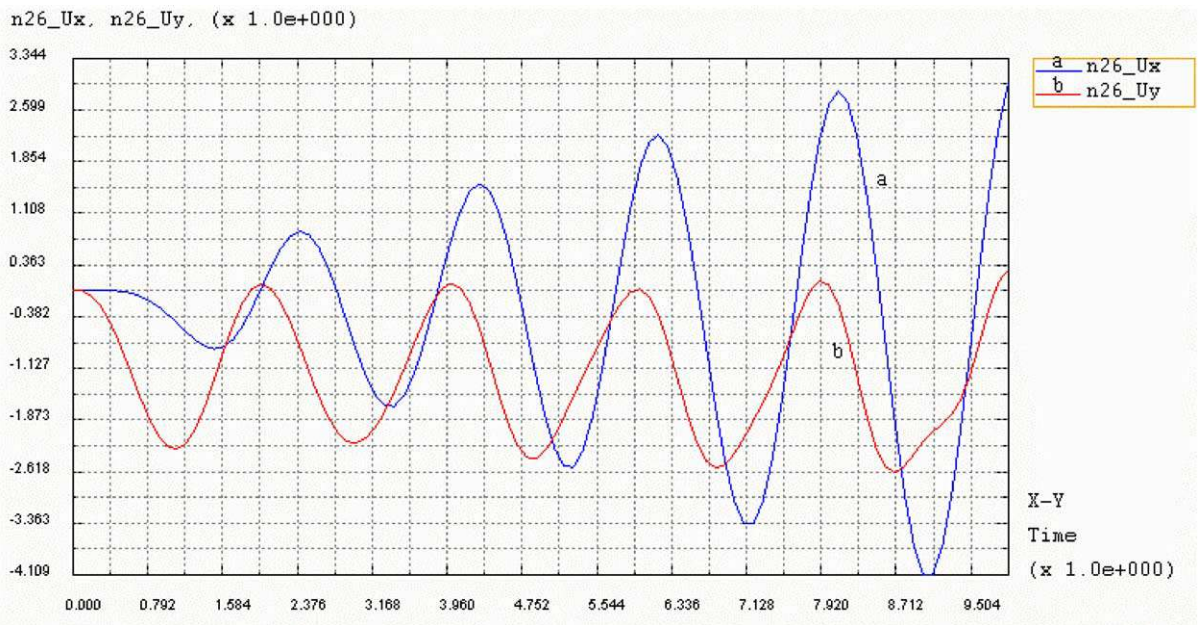

Fig. 11. Displacements (Ux and Uy) of sphere versus time.

5. Conclusions

In this article, we have presented a practical application of the object-oriented approach to finite element analysis and the software prototype FER/Mech. The open architecture of the program facilitates further developments and adapts to suit specific needs easily and quickly. Moreover, the proposed user interface has proved to be satisfactory and flexible. We have seen that absolute natural coordinates to model rigid bodies is an efficient way to respect the concept of modularity of OOP in FEM. Our experience shows that C++ offers serious benefits for scientific computing. The authors feel confident that OOP in C++ will promote the development of computational tools for structural analysis and GUI applications.

References

[1] Jacobsen KP. Fully integrated superelements: a database approach to finite element analysis. Comput Struct 1983;16:307–15.

[2] Rajan SD, Bhatti MA. Data management in FEM-based optimization software. Comput Struct 1983;16:317–25.

[3] Murthy TS, Shyy YK, Arora JS. MIDAS: management of information for design and analysis of systems. Adv Engng Software 1986;8: 149–58.

[4] Kunz DL, Hopkins AS. Structured data in structural analysis software. Comput Struct 1987;26:965–78.

[5] De Figueiredo LH, Gattass M. A database management system for efficient storage of structural loading. Comput Struct 1989;32:1025–34.

[6] Wang S. A conception of module library and data base management system for finite element analysis. Comput Struct 1989;26(1): 073–1083.

[7] Touzot G. S.I.C.1.1: Réflexion sur l'architecture des logiciels de modélisation. Rapport interne, Université de Technologie de Compiègne, France; 1986.

[8] De Saxcé G. Le projet CHARLY: un logiciel de calcul par elements finis et elements frontiers de seconde generation. Séminaire de genie logiciel, Division MSM, Université de Liège, Belgium; 1987.

[9] Verpeaux P, Charras T, Millard A. CASTEM 2000: une approche moderne du calcul des structures. In: Fouet JM, Ladevèze P, Ohayon R, editors. Proc. Calcul des structures et intelligence artificielle, Pluraris; 1988.

[10] Forde BWR, Foschi RO, Stiemer SF. Object-oriented finite element analysis. Comput Struct 1990;34(3):355–74.

[11] Filho JSRA, Devloo PRB. Object-oriented programming in scientific computations: the beginning of a new era. Engng Comput 1991;8: 81–7.

[12] Mackie RI. Object-oriented programming of the finite element method. Int J Numer Meth Engng 1992;35(2):425–36.

[13] Pidaparti RMV, Hudli AV. Dynamic analysis of structures using object-oriented techniques. Comput Struct 1993;49(1):149–56.

[14] Raphael B, Krishnamoorthy CS. Automating finite element development using object-oriented techniques. Engng Comput 1993;10: 267–78.

[15] Scholz SP. Elements of an object-oriented FEM++ program in C++. Comput Struct 1992;43(3):517–29.

[16] Zeglinski GW, Han RPS, Aitchison P. Object-oriented matrix classes for use in a finite element code using C++. Int J Numer Meth Engng 1994;37(22):3921–37.

[17] Lu J, White DW, Chen WF, Dunsmore HE. A matrix class library in C++ for structural engineering computing. Comput Struct 1995; 55(1):95–111.

[18] Zimmermann T, Duboispelerin Y, Bomme P. Object-oriented finite element programming. 1. Governing principles. Comput Meth Appl Mech Engng 1992;98(2):291–303.

[19] Duboispelerin Y, Zimmermann T, Bomme P. Object-oriented finite element programming, 2. A prototype program in Smalltalk. Comput Meth Appl Mech Engng 1992;98(3):361–97.

[20] Duboispelerin Y, Zimmermann T. Object-oriented finite element programming, 3. An efficient implementation in C++. Comput Meth Appl Mech Engng 1993;108(1–2):165–83.

[21] Menetrey P, Zimmermann T. Object-oriented non-linear finite element analysis—application to J2 plasticity. Comput Struct 1993; 49(5):767–77.

[22] http://www.ann.jussieu.fr/~pironneau/freefem.htm

[23] Aazizou K, Besson J, Gailletaux G, Hourlier F. Une approche C++ du calcul par éléments finis. Colloque National en Calcul des Structures, 11–14 mai Giens, France 1993;2:709–22.

[24] Feng ZQ, Aazizou K, Hourlier F. 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 1993;2:1141–56.

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

[26] Gao S, Wan H, Peng Q. An approach to solid modeling in a semi-immersive virtual environment. Comput Graph 2000;24:191–202.

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

[28] Feng ZG, Feng ZQ, Domaszewski M. FER/View: un post-processeur général de calcul par éléments finis. 4ème Colloque National en Calcul des Structures, Edition Teknea, Giens, France 18–21 Mai 1999;2:883–87.

[29] Geradin M, Cardona A. Flexible multibody dynamics: a finite element approach. New York: Wiley; 2001.

[30] Newmark NM. A method of computation for structural dynamics. ASCE J Engng Mech Div 1959;85:67–94.

[31] Feng ZQ, Feng ZG, Domaszewski M. Some computational aspects for analysis of low and high-velocity impact of deformable bodies. Int J Non-Linear Mech 2002;37(6):1029–36.

[32] Joli P, Pascal M, Gilbert JR. Numerical simulation of multibody systems with time dependent structure. The 14th ASME Biennal Conference on Mechanical Vibration and Noise, Albuquerque, USA, September 19–22, 1993.

[33] Gear CW, Petzold LR. ODE methods for the solution of differential/ algebraic system. SIAM J Numer Anal 1984;21:716–28.

[34] Baumgarte J. Stabilization of constraints and integrals of motion in dynamical systems. Comput Meth Appl Mech Engng 1972;1:1–16.

[35] Fahrat C, Crivelli L, Géradin M. On the spectral stability of time integration algorithms for a class of constrained dynamics problems. 34th AIAA adaptative structure forum, LaJolla, CA; April 19–22, 1993.

[36] Bayo E, Garcia de Jalon J, e Serna MA. A modified Lagrangian formulation for the dynamic analysis of constrained mechanical systems. Comp Meth Appl Mech Engng 1988;71:183–95.

[37] Wehage R, Haug EJ. Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic system. ASME J Mech Des 1982;104:245–55.

[38] Trahn DM. Equations of motion of multibody systems in the ESA-MIDAS software. International Conference on Spacecraft Structures and Mechanical Testing ESTEC, Noordwijk; April 24–26 1991.

[39] Bae DS, Haug EJ. A recursive formulation for constrained mechanical system dynamics, part I: open loop systems. Mech Struct Mach 1987; 359–82.

[40] Featherstone R. Robot dynamics algorithms. Dordrecht: Kluwer Academic Publishers; 1987.

[41] Feng ZQ, Touzot G. Analysis of two and three dimensional contact problems with friction by a mixed finite element method. Revue Européenne des Eléments Finis 1992;1(4):441–59.

[42] Wright JrRS, Sweet M. OpenGL superbible: the complete guide to OpenGL programming for Windows NT and Windows 95. Waite Group Press; 1996.