State-of-the-art Programming Techniques of Finite Element Methods for Electromagnetic Field Computation

S. L. Ho[1], Shuangxia Niu[1], W. N. Fu[1], and Jianguo Zhu[2]
[1]Department of Electrical Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong
[2]Faculty of Engineering, University of Technology, Sydney, P.O. Box 123, Broadway NSW 2007, Australia
eeslho@polyu.edu.hk

Abstract — State-of-the-art programming techniques of finite element methods (FEMs) for electromagnetic field computation are presented. It covers program structure, data structure and equation solver. The advantages of the program structure are that multi-developers can work on different solvers and share common algorithms. The proposed data structure allows two dimensional (2-D) FEM, multi-slice FEM and three-dimensional (3-D) FEM can share the same data structure; it is also efficient for organizing FEM programs, convenient for mesh generation and motion problems and allows quick access to all data. The equation solver can automatically deal with Dirichlet boundary conditions, master-salve boundary conditions and all other constraints in sparse matrix equations. The sub-matrix operation technique allows that electric circuit equations can be easily coupled with electromagnetic field equations.

I. INTRODUCTION

To develop a FEM program, a proper data structure is the foremost important issue to be considered [1-2]. In object-oriented programming, the data structure is the foundation of all algorithms. In FEMs of many electric devices, such as electric machines, 2-D model, multi-slice model or 3-D model can be used [3]. They simulate the same problem in different levels. The multi-slice and 3-D models have high accuracy but need long computing time. It is usually a good practice to use the 2-D model first to get a rough solution, followed by using multi-slice and 3-D models to consider effects due to skewed slots and end-windings. In this paper a new data structure based on C++ object-oriented programming technique is presented. The 2-D, multi-slice and 3-D models can use the same data structure. Hence it is convenient to switch among these three models when simulating the same problem. The source codes of the three models share the same data structure. The geometry has a nested structure so it is easy to deal with holes inside the geometry. The mesh is the property of the face and volume objects. The mesh generation starts from the most internal object, first on faces, then on volume objects [4]. It is convenient to deal with mechanical motion problems as the motion is the property of the volume object. It also provides a platform to allow the data to be accessed quickly. In the paper a program structure and an equation solver are also presented.

II. PROGRAMMING TECHNIQUES

A. Program Structure

The FEM program is composed of pre-processing, the solvers and post-processing modules. The solvers have the functions of meshing, pre-assembly, assembly, matrix equation solver. The program adopts object-oriented technology, and inheritance is one of the main concepts of the program. The base class defines the interface, and the derived one provides the implementation specific to this derived class [5]. Virtual functions can be used to define a set of interfaces for the base class. It is convenient to share the codes and maintain, extend the software efficiently. Template is another technology to express the commonality. With it the programmer can regenerate the entire family of related classes [5]. With the object-oriented technology, the program structure allows multi-programmers to work on different solvers and share common algorithms.

For this proposed FEM program, the base class of the solver is named as class **FeSolverBase**. All other solvers should be derived from it. The sizes of the included data items in the solver, such as **FeData**, **FeVertex**, **FeFace and etc.** are not known before hand, so it is convenient to organize them in a dynamic data structure and a template class is used here.

In each the base class, if necessary, it has the functions of reading data and writing data. Each solver can derive its own classes from the base classes and the functions of reading data and writing data are always shared by different solvers. Therefore the input and output data formats can be kept the same.

The derived solvers are **FeSolver2d**, **FeSolverMs, FeSolver3d**, and they are developed by different programmers and can used to solve the 2-D, multi-slice and 3-D problems conveniently.

B. Data Structure

The data of a FEM project before meshing is all defined in the class **FeData**. The **FeData** has its own methods of reading and writing, so it is easy to pass the data among the pre-processor, the solvers and the post-processor. All data file can use standard and structured Extensible Markup Language (XML) so it is convenient to communicate with other software. The main data members of **FeData** are shown in Fig. 1.

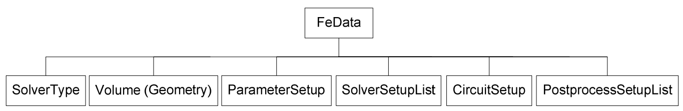

Fig. 1. The class **FeData**

Besides the geometry information, the class **FeData** also contains solver setup information, motor and circuit data which are necessary in electromagnetic computing and post processing. Another advantage of this class is that it integrates the data of 2-D FEM, or multi-slice FEM or 3-D FEM into one class.

The class **Volume** contains the geometry data as shown in Fig. 2. It has a nested structure. The top is the background object. It contains the face list of the external boundaries, the volume list inside the background object, etc. Each volume object in the internal volume list is also an instance of the class **Volume**, so that the background object and all internal volume objects have the same data structure and are consisting of a nested structure. Each face has a pointer which points to the volume object to which it belongs. So it is quick to find the face from the volume object, or find the volume object from the face.

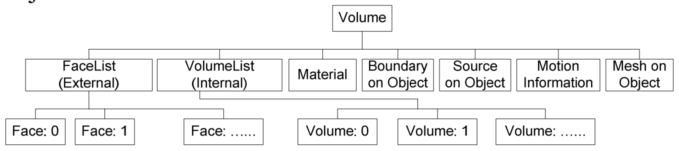

Fig. 2. The class **Volume**

The class **Face** is shown in Fig. 3. It contains an edge list of the external boundaries. For the 2-D FEM and multi-slice FEM, the face is not limited to triangle, it can be straight line, triangle, square or polygon. For each face we define the external edges and internal face. In the 3-D model the class **Face** may contain a list of internal faces. Each face in the face list is also an instance of the class **Face**, so that all faces have the same data structure.

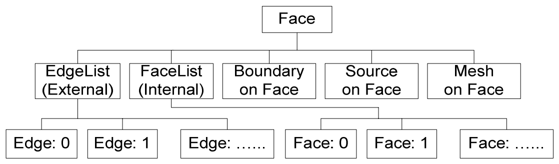

Fig. 3. The class **Face**

The class **Edge** is shown in Fig. 4. It contains a vertex list. For the 2-D model the vertex list only has one vertex; for multi-slice model, the vertex list only has two vertexes.

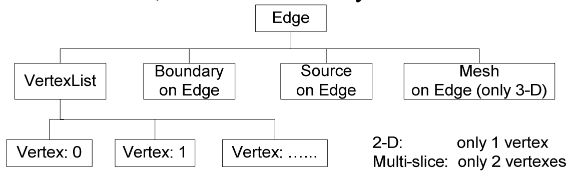

Fig. 4. The class **Edge**

The relationship of the main classes is demonstrated in Fig. 5. The solid line means one class's dependence on another class. The dash line means the dependence and access of one class to another. In our design the material class is related with Volume and Face. Inciting Source is connected with Volume, Face and Edge. Volume, Face, Edge and Vertex all can be set as Boundary condition.

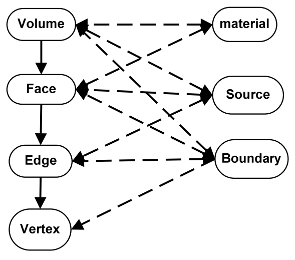

Fig. 5. The logic relationship of the data

Another important class is the **FeMeshData** which is created from FeData or imported from mesh file. Class FeData is general enough to take into consideration a great number of element types such as triangle or quadrilateral for 2-D programs, cubes or tetrahedrons for 3-D programs. It has similar structure to that of FeData. But it contains all mesh information necessary for equation solving. The mesh generation uses refining quality, optimal meshing methods and considers all possible condition such as holes, boundaries, multi-domains.

The class **FeSolver** is the top class which contains data members of class FeData and FeMeshData. The classes of the 2-D, multi-slice and 3-D solvers are derived from the base-class **FeSolver** so different solvers can be developed independently but they share the same data structure. More details of the data structure will be given in the full paper.

C. Equation Solver

The equation solver can solve symmetric, asymmetric, complex-valued sparse-matrix equations. It can automatically deal with Dirichlet boundary conditions, master-salve boundary conditions and all other constraints in sparse matrix equations. The sparse matrix storage technique allows the contributions can be added to the matrix any time. It does not need to pre-determine the positions of non-zero elements. It supports many matrix operations. Sub-matrixes can be added into the main coefficient matrix conveniently which allows that electric circuit equations can be easily coupled with electromagnetic field equations.

III. EXAMPLES

The developed programs have been widely applied to design many electric devices. Fig. 6 shows the computed magnetic flux density of a nodal flux-modulated permanent magnet motor developed by the authors.

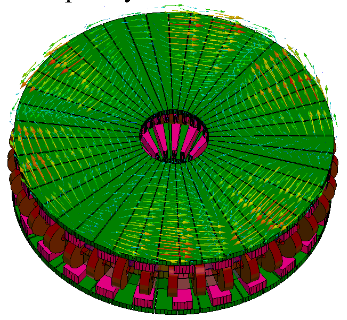

Fig. 6. Magnetic flux density on the surface of the rotor iron core

IV. REFERENCES

[1] A. Promwungkwa, "Data structure and error estimation for an adaptive p-version finite element method in 2-D and 3-D solids", Ph.D Dissertation, 1998.

[2] H. Karutz, W. B. Kraetzig , "A quadtree data structure for the coupled finite-element/element-free Galerkin method", *International Journal for Numerical Methods in Engineering*, vol. 53, no.2, pp. 375-391, 2001.

[3] W. N. Fu, S. L. Ho, H. L. Li and H. C. Wong, "A multislice coupled finite-element method with uneven slice length division for the simulation study of electric machines," *IEEE Trans. Magn.*, vol. 39, no. 3, pp. 1566-1569, May 2003.

[4] V. Chugunov, Yu. Vassilevski, "Parallel multilevel data structures for a nonconforming finite element problem on unstructured meshes," *Russian Journal of Numerical Analysis and Mathematical Modelling*, vol. 18, no. 1, pp. 1-11, Feb. 2003.

[5] S. Kumar, "Object-oriented finite element programming for engineering analysis in C++", *Journal of Software*, vol. 5, no. 7, pp. 689-696, Jul 2010.