Advances in Engineering Software 76 (2014) 133–147

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

ESFM: An Essential Software Framework for Meshfree Methods

Yo-Ming Hsieh, Mao-Sen Pan *

Department of Construction Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan

ARTICLE INFO

Article history:
Received 6 November 2013
Received in revised form 31 May 2014
Accepted 16 June 2014
Available online 11 July 2014

Keywords:
Meshfree methods
Meshless methods
Software framework
Object-oriented design
Design pattern
Weak-form

ABSTRACT

This paper describes an Essential Software Framework for Meshfree Methods (ESFM). Through thorough analyses of many existing meshfree methods, their common elements and procedures are identified, and a general procedure is formulated into ESFM that can facilitate their implementations and accelerate new developments in meshfree methods. ESFM also modulates performance-critical components such as neighbor-point searching, sparse-matrix storage, and sparse-matrix solver enabling developed meshfree analysis programs to achieve high-performance. ESFM currently consists of 21 groups of classes and 94 subclasses, and more algorithms can be easily incorporated into ESFM. Finally, ESFM provides a common ground to compare various meshfree methods, enabling detailed analyses of performance characteristics.

1. Introduction

Over the past two decades, the meshfree method is an active and popular research topic in many fields. Due to their meshless nature, meshfree methods can yield more accurate solutions than FEM (finite element methods), and they have other advantages [1–3] over FEM such as: (1) avoiding the manual effort in designing appropriate FE (finite element) meshes, (2) evading element distortion issues seen in large deformation problems [4–9], and (3) simulating crack propagations with ease [9–14]. Many literatures on meshfree methods have been published, and some important overview or comparison papers have been organized and discussed [15–19]. In contrast, very few studies have discussed implementations of meshfree methods. Most open literatures associated with implementation issues of meshfree methods focus on the construction of shape functions [20,21], and some discusses one particular flavor of meshfree methods [17,22,23]. Some open source or public domain codes for meshfree or related methods [24–31] were developed to demonstrate new procedures or new methods such as IGA (isogeometric analysis) using Matlab or Octave. Using these software packages for developing meshfree methods is convenient and keeps researchers focus on developing the meshfree method from mathematical perspectives. However, it becomes impossible to explore how data structures (such as sparse-matrix storage) can affect performance of developed meshfree analyses. Furthermore, most current implementations of meshfree methods do not necessarily consider best practices in software engineering, such as design patterns [32] and code reuse. No publication to date, to author's knowledge, discusses software framework for meshfree methods developed in C^{++}. There are several advantages using C^{++} for development. First, programs developed in native languages such as C^{++} typically perform much faster than those developed in interpreted languages do. Furthermore, many parallel computing toolkits or technologies such as OpenMP [33], CUDA [34], Intel Thread Building Blocks [35], and Thrust [36], are only available to C and C^{++}. Therefore, using C^{++} for development enables the use of these parallel computing toolkits or libraries.

This paper presents an Essential Software Framework for Meshfree Methods (ESFM), which supports implementing many basic meshfree methods and their associated numerical algorithms. It is intended for ESFM to solve partial differential equations raised from various physical problems, such as solid mechanics, and heat conduction, using different flavors of meshfree methods. As a result, many meshfree methods, numerical methods, and formulation methods need to be considered into ESFM. We believe ESFM provides a logical and well-organized framework to fulfill the aforementioned goals, and it is the first of its kind in the context of meshfree methods, while many other software frameworks were developed for finite element method [37–57] or for discrete element method [58,59].

It is believed the ESFM cannot only facilitate the development of programs using meshfree methods, but also stimulate new research and development efforts in this field. During the development of ESFM, extensive use of (1) object-oriented analysis and

* Corresponding author. Tel.: +886 227333141.
E-mail addresses: ymhsieh@mail.ntust.edu.tw (Y.-M. Hsieh), d9505503@mail.ntust.edu.tw (M.-S. Pan).

http://dx.doi.org/10.1016/j.advengsoft.2014.06.006
0965-9978/© 2014 Elsevier Ltd. All rights reserved.

design (OOAD) and (2) well-known design patterns [32] allows ESFM inherit benefits of being extensible and maintainable from object oriented programming (OOP). Furthermore, with carefully structured class hierarchy, new development can be achieved with less coding effort by reusing most of the developed classes. Furthermore, the framework is validated by implementing a program for solving plane-stress beam problem well documented in most literatures using three different meshfree methods.

The rest of this paper is organized as the following: Section 2 analyzes the requirements for ESFM and identifies its basic components. Section 3 presents the design of ESFM. Section 4 describes three major procedures and operations in most meshfree methods and their implementations in the current ESFM. Section 5 demonstrates sample calculations on two-dimensional and three-dimensional problems with different shape functions, sparse matrix storage schemes, and parallelization. Conclusive summary are given in Section 6.

2. Common procedures and components in weak-form meshfree methods

ESFM was designed and developed based on the following three general requirements. (1) It should be able to implement most existing meshfree methods. (2) It should provide adequate modulization and abstraction to allow independent developments of algorithms in various parts of meshfree methods with minimal efforts. For example, implementations of search algorithms should not affect implementations in shape functions of meshfree methods. (3) Performance should also be considered in order to develop high-performing meshfree analysis programs. Furthermore, we only focus on the most fundamental aspects of meshfree methods that are common to all surveyed methods. Therefore, several features in surveyed literatures such as adaptivity, crack growth, and multi-physics are excluded.

Based on these requirements, it is necessary to review existing meshfree methods to: (1) identify common procedures amongst various meshfree methods for unifying implementation efforts; (2) discover fundamental components in meshfree methods; and (3) isolate performance-critical numerical method components.

2.1. Common procedures for weak-form meshfree methods

By analyzing calculation procedures in some major meshfree methods [60–63] and finite element methods, it is not difficult to

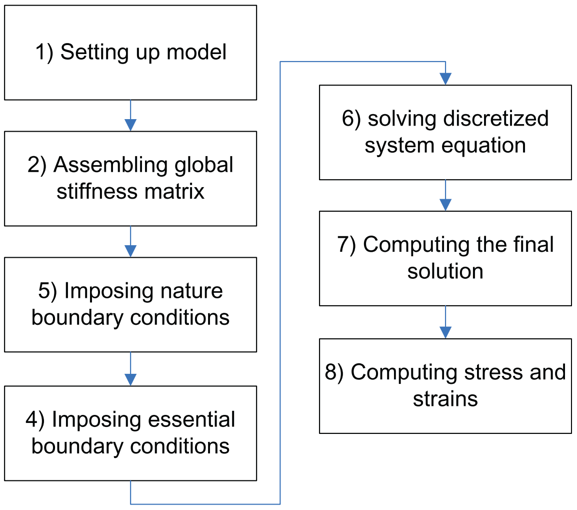
1) Setting up model
2) Assembling global stiffness matrix
5) Imposing nature boundary conditions
4) Imposing essential boundary conditions
6) solving discretized system equation
7) Computing the final solution
8) Computing stress and strains

Fig. 1. The flowchart for the identified calculation procedure for general meshfree analyses.

identify a unified calculation procedure for these methods. Fig. 1 shows the flowchart for the identified calculation procedure for general meshfree analyses in the context of linear and static solid mechanics, but this can be easily extended to solving problems formulated from other fields as well. These general steps are:

1. *Setting up model*: the model for meshfree analyses mainly consists of field nodes, boundary nodes, integration meshes needed by weak formulation, and material constitutive models. Field nodes are used by shape functions (or approximation functions) for generating approximations to the problem being solved, and often evaluations of derivatives are required. Boundary nodes are used to impose boundary conditions. Integration meshes can be manually assigned or automated generated by Delaunay or Voronoi diagram [64] in 2D, but robust-generation of 3D integration meshes remains difficult. Material constitutive models are mainly used to consider material behaviors during evaluation of global stiffness matrices in later steps.
2. *Assembling global stiffness matrix*: the governing equation is first approximated using a chosen meshfree approximation method, and then integrated over the entire problem domain using numerical integration methods such as Gauss integration [65] on integration points (which are different from field nodes), or nodal integration [66] using field nodes.
3. *Imposing natural boundary conditions*: imposing natural boundary conditions such as external loadings and body forces is similar to the procedures in finite element methods. Natural boundary conditions often contribute to the load vector, and the global stiffness matrix is usually untouched.
4. *Imposing essential boundary conditions*: the imposition of essential boundary conditions can be difficult in meshfree methods. There are two types of shape-functions, one with Kronecker delta property and one without. Those without the Kronecker delta property require special treatments on imposing essential boundary conditions. Often imposing essential boundary conditions leads to modifications on both global stiffness matrices and load vectors.
5. *Solving discretized system equation*: once the previous steps are completed, a discretized system of linear equations $Ku = F$ is formed, and the system of equations can be then solved using various methods.
6. *Computing the final solution*: for meshfree methods using shape functions without the Kronecker delta property, the computed u from the previous step needs to be substitute into the shape function in order to recover the true solution. This step is unnecessary for shape functions that possess the Kronecker delta property.
7. *Computing stresses and strains*: after the final solution is obtained, strains and stresses in each gauss point can then be calculated by using shape functions and supplied constitutive laws.

The above steps do not consider nonlinearity solution procedures such as Newton–Raphson iterations and use linear-static solid mechanics notations (K, u, and F). We focus mainly on linear-static solid mechanics procedure in this study.

2.2. Fundamental components for weak-form meshfree methods

To ease the implementations of meshfree analysis programs by code re-using and to encourage new developments in meshfree methods, it is necessary to categorize and modulize major components in meshfree analyses, so that various algorithms in meshfree analyses can be independently developed without being affected or affecting other already developed modules. Therefore, not only