An object-oriented framework for finite element analysis based on a compact topological data structure

Lauren L. Beghini a, Anderson Pereira b, Rodrigo Espinha b, Ivan F.M. Menezes b, Waldemar Celes b, Glaucio H. Paulino a,⇑

aDepartment of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, USA
bTecgraf, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil

Abstract

This paper describes an ongoing work in the development of a finite element analysis system, called TopFEM, based on the compact topological data structure, TopS [1,2]. This new framework was written to take advantage of the topological data structure together with object-oriented programming concepts to handle a variety of finite element problems, spanning from fracture mechanics to topology optimization, in an efficient, but generic fashion. The class organization of the TopFEM system is described and discussed within the context of other frameworks in the literature that share similar ideas, such as GetFEM++, deal.II, FEMOOP and OpenSees. Numerical examples are given to illustrate the capabilities of TopS attached to a finite element framework in the context of fracture mechanics and to establish a benchmark with other implementations that do not make use of a topological data structure.

1. Introduction

In the past few decades, the presence of the finite element method has become more important in both research and industry, due to its ability to analyze and study detailed information that other tools cannot provide [3,4]. This growth has left practicing engineers with a steadily increasing need for more computational power. With today’s technology, designs continue to grow larger and faster, resulting in finite element models containing millions of elements that must be handled in an efficient and timely fashion. Furthermore, as computer hardware continues to advance, there is a necessity for software to grow accordingly to make the process capable of handling such large problems in the fields of solid, structural and fluid mechanics.

Object-oriented programming has become more common for computationally intensive finite element applications spanning the field of continuum mechanics (e.g. aeronautical, automotive, biomechanical industries) due to its flexibility from the concepts of inheritance, polymorphism and encapsulation; however, the efficiency of these programs can be impacted significantly by the underlying data structures. In this paper, we introduce a new finite element analysis program (TopFEM) based on the concepts of object-oriented programming in conjunction with the topological data structure to create a robust framework for adaptive finite element problems. TopFEM was implemented in a way such that (i) the data structure (TopS) stores only node and element data explicitly, with other topological entries implicitly represented (i.e. they are only retrieved when needed) and (ii) expansion of the program is straightforward, as in other frameworks.

1.1. The topological data structure (TopS)

Typical data structures consist of element-node mesh representations with tables to store the node and element information using the node connectivity information. While this data structure is seemingly simple and easy to implement, often it suffers in terms of efficiently providing the necessary adjacency information required to solve several problems (e.g. fragmentation simulation, visualization techniques) [5–9]. The topological data structure (TopS) [1,2] instead contains a complete and compact data structure which utilizes a relatively small amount of memory, while still providing the user with access to all topological entities. Furthermore, TopS is naturally applicable to adaptive meshes.

TopS uses abstract topological entities (facet, edge, vertex) to represent a finite element mesh. With large models, the amount of data can require extensive storage space if not carefully considered, often rendering the cost of large models prohibitive. Moreover, the access to each topological entity can be computationally intensive, and relatively complex, making it difficult for the user to edit or query the data efficiently. Thus, the topological data structure selected here is well suited for the problems we aim to solve due to its (i) compactness, (ii) completeness and (iii) adaptivity.
The data structure used in this work is compact in comparison with other data structures in the sense that the storage space requirement is reduced, but topological information can be retrieved in constant time, or time proportional to the output size \[1,2\]. The rationale behind the reduced storage space can be explained by the fact that only nodes and elements are explicitly stored in internal arrays, using a relatively small amount of memory. In finite element meshes, the node and element arrays are the most important entities in the respect that they can be used alone to construct a finite element model; therefore, the node and element entities are selected to be stored explicitly. In many finite element representations, the nodal array typically stores the \(x,y,z\) position in space for each node, and the element array holds connectivity information. Conversely, in the topological data structure, TopS, the nodal array is modified to store both position and a connectivity information. In many finite element meshes, entities are selected to be stored explicitly. In many finite element applications for efficiency, organization and reusability in parallel and distributed computing environments, where each computer has its own local private memory with access to the memory of other machines to exchange information when numerical simulations are prohibitively expensive for a single processor.

The use of object-oriented programming is especially suited for parallel and distributed computing applications due to its ability to easily incorporate modifications in data structures for parallel/distributed computing, such as abstractions for data mapping, communication and message passing \[3\]. Thus, the TopFEM framework is flexible enough to be used across many different computer platforms.

To incorporate such object-oriented programs into parallel computing environments, many modern libraries have been developed, such as ParTopS \[14\] for dealing with the data structure, and PetSC \[15–17\], PARDISO \[18,19\], and AMD Core Math Library (ACML) \[20\] for parallel linear algebra (linear solvers), among others. Such libraries can be easily implemented as extensions of the current object-oriented framework, however, some challenges result, including decomposing computations between processors, dynamic mesh partitioning (load balancing among processors), efficient communication between partitions, and adaption of current serial algorithms for parallel execution. For a discussion on these issues, the interested reader can refer to references \[21–23\]. For linear systems of equations, TopFEM uses a base class\(\text{de}\)-\(\text{class}\) interface with these libraries to extend them to parallel\(\text{distributed}\) computing in a manner similar to the \text{Actor Model} of OpenSees, in which the \text{Actor abstract base class} provides methods for descendent classes to communicate with their \text{Shadow objects} \[24\].

1.3. Paper organization

The remainder of this paper is organized as follows: Section 2 reviews the state-of-the-art in object-oriented programming and discusses these concepts in the context of finite element programming. Next, in Section 3, the TopFEM class organization and framework are presented. Numerical examples are given to illustrate the effectiveness of the new framework using the topological data structure (compared to previous implementations for fracture mechanics) in Section 4. Finally, we conclude with some remarks in Section 5.

2. Object-oriented finite element concepts

Significant research has been conducted to demonstrate the effectiveness of finite element applications based on object-oriented programming. A detailed discussion on incorporating finite element concepts in an object-oriented framework is included in the series of papers by Zimmermann et al. \[25–31\], where they show that object-oriented programming can be used to achieve appropriate levels in development speed, ease of maintenance, software reliability and reusability. We discuss the main contributions of an object-oriented framework for finite element programming next.

2.1. Related work

Object-oriented programming has been incorporated into finite element applications for efficiency, organization and reusability in previous work. Here, we give a brief review of the existent work that has been presented on object-oriented finite element programming.

One of the first implementations of finite element programs using object-oriented concepts was presented in a series of papers by Zimmermann and Dubois-Pèlerin \[25–27\]. In this early work, they demonstrate the numerical efficiency of using the C++
programming language (in comparison to Fortran), and its flexibility and capabilities to interface with the heavily used C-language.

The finite element framework using object-oriented programming developed by Rucki and Miller [32] provided "new levels of computational flexibility for general finite element-based structural modeling" to use classes to define the finite element modeling abstractions. This framework was developed to support a wide range of algorithmic methods and techniques which included degrees of freedom and element-based, iterative and direct algorithms for nonlinear problems in static and dynamic contexts.

These concepts were explored further in the work of McKenna [24], known commonly nowadays in the engineering community as OpenSees (Open System for Earthquake Engineering Simulation), or previously as G3 [33]. This work emphasized the concept of breaking computations down into tasks, that are then assigned to processes. These tasks were especially suited for object-oriented programming due to the intrinsic nature of tasks as the invocation of object methods, where the tasks share common data that could be easily managed in this framework.

Another piece of work initiated in the early 1990s was FEMOOP [34], which provided an object-oriented framework for finite element problems. Using two OOP classes, Analysis Model and Shape, FEMOOP was developed to handle multidimensional models and associated natural boundary conditions in a generic fashion. Similar to TopFEM, for example, the Element class has references to the Shape, Analysis Model, and Integration Point classes, which define generic behaviors. For instance, the Shape class contains geometric and field interpolation information, such as the dimension, shape, and number of nodes. The Analysis Model defines the behavior for the type of analysis to be performed (e.g. plane stress, plane strain, solid, 3D beam/truss, etc.). The Integration Point class or FemIntegration class in TopFEM, holds information about the integration order, number of points, coordinates, weights, etc. Thus, for each element, a method to compute the stiffness matrix is independent of the specific shape (quadrilateral, hexahedral, etc.), dimension (1D, 2D and 3D), and interpolation order (linear, quadratic, and so on).

Similarly, the GetFEM++ (Generic Toolbox for Finite Element Methods in C++) library [35] also features generic management of meshes (i.e. arbitrary geometry, arbitrary dimensions), some generic assembling methods, and interpolation methods. Instead of using the topology to describe a finite element model as TopFEM does, the GetFEM++ framework describes models in the form of bricks, with the objective of high reusability of the code. The system of bricks is used to assemble components for a variety of model types (small/large deformations, Helmholtz, elliptic, etc.), components representing boundary conditions and components representing constraints [36].

Another finite element program library, known as deal.II (A Finite Element Differential Equations Library) [37], is based on the data structure, p4est [38]. The p4est data structure, like TopS/ParTopS [14], is implemented to adaptively refine and coarsen meshes for parallel computing, as in the case of fracture simulations. However, while TopS/ParTopS uses the connectivity and topological information for adaptivity and coarsening, p4est uses a forest of two-dimensional quadtree or three-dimensional octree.

A related framework, referred to as NLS++, was presented in [39]. In this work, the use of object-oriented programming is demonstrated for implementation of a unified library of several nonlinear solution schemes in finite element programming. Because the solvers share a common interface, object-oriented programming was crucial to establish the class structure within this framework (consisting of Model, Control and Linear Solver classes).

In the TopFEM framework presented here, we propose a new object-oriented program using the concepts of the topological data structure and object-oriented finite element programming. The topological data structure was selected to efficiently represent finite elements of any type, while utilizing the concept of oriented entities to significantly reduce the storage space requirements. The object-oriented finite element programming portion was inspired by FEMOOP [34] and OpenSees [24] as described above, due to their clear organization and flexibility.

2.2. Object-oriented programming for finite element applications

In finite element applications based on object-oriented programming, the objects are broken down in order to provide flexibility in the creation of different analyses [24,27,34]. This process is performed by identifying the main tasks to be completed in an analysis, abstracting them into separate classes, and specifying an interface to facilitate the interaction between classes and allow for straight-forward introduction of new classes [33]. For example, in the class organization implemented in OpenSees, the creation of the Analysis class as an aggregation can only be performed after the Integrator, SolnAlgorithm, Constraint Handler, and so on classes are created. This concept is important in that small objects can be used in aggregation to compose larger objects for performing a task.

Another essential concept involved in object-oriented finite element applications is the use of different levels. For example, the analysis and the element levels are independent so that they are easily extendable, but can communicate with each other. The communication of these two levels is made in two directions, for example, the Analysis class will need to update the Domain class at some point, and vice versa. This communication is important for future developments in the parallel/distributed field, for example.

In the next section, we will discuss the significance and application of these two main ideas within the TopFEM framework, including a detailed description of the interaction between classes and the hierarchical structure of the program.

3. The TopFEM system

The TopFEM system is a finite element analysis framework based on the TopS [1,2,14] topological data structure. By using a complete and compact data structure, analysis codes can take advantage of efficient topological queries and mesh modification operations required by some classes of applications, e.g. fracture propagation [12,40] and adaptive simulations. One of the main goals of the TopFEM system is to provide a common basis onto which independent finite element analysis codes can be consistently integrated. While still in an early stage of development, the current implementation of the TopFEM system has been successfully applied to the solution of a variety of problems, as discussed in Section 4. The basic class organization of the system is briefly presented next.

Fig. 3.1 shows an illustration of the high-level classes of the TopFEM system. The FemDomain class (Fig. 3.1) is a container responsible for storing the finite element mesh and associated model attributes, such as materials, boundary conditions and load objects, which can be specified using the FemModelBuilder class. Although in the context of traditional FEM the only data structures required for representing the finite element mesh are a table of nodes and a table of elements, in TopFEM, however, the mesh is represented by a TopModel object, provided by the TopD data structure. As a result, efficient access to the complete set of topological relationships is made available to the analysis application without significant additional memory and performance penalties compared with the simple mesh representation.
Per-element and per-node attributes are represented by the FemNode and FemElement classes (Fig. 3.1), which are associated to the corresponding topological entities in the finite element mesh, TopNode and TopElement. The TopNode entity and its associated attribute set (FemNode) correspond to a discrete point in the finite element domain for which response quantities are defined. The function of TopNode is to store the geometric coordinates of the point, while FemNode is responsible for the response quantities (e.g. displacement, velocity and acceleration). Each TopElement is connected to a number of TopNodes. Topological information such as the number of nodes of an element and node identifiers are obtained from the element through the interface of the TopModel class. The FemElement object associated to a TopElement entity is responsible for computing analysis-specific element quantities, such as stiffness and mass matrices. The FemElement class represents the finite elements and is extended by the subclasses FemParametricElement and FemBeamElement, which represent, respectively, the parametric finite elements and the beam elements, as shown in Fig. 3.2.

The separation of concerns between mesh representation and analysis attributes (the finite element domain) allows either the mesh representation or analysis attributes to be replaced or
adapted regardless of each other, in order to suit different application needs. Specific node and element data and behavior required by an analysis application can be modeled through specialized sub-classes of FemNode and FemElement, respectively. This allows the TopFEM framework to be extended with new types of nodes and elements. Nodal and element attributes are efficiently stored in dy-

Table 1
Performance of building analysis.

<table>
<thead>
<tr>
<th>Software</th>
<th>Solver</th>
<th>Elapsed time (s)</th>
<th>Max. displacement at node 4062 (m, rad)</th>
<th>Dx</th>
<th>Dy</th>
<th>Dz</th>
<th>Rz</th>
<th>Ry</th>
<th>Rz</th>
</tr>
</thead>
<tbody>
<tr>
<td>TopFEM</td>
<td>UMFPACK</td>
<td>0.548</td>
<td></td>
<td>0.7613</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002015</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>CROUT</td>
<td>1.690</td>
<td></td>
<td>0.7613</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002015</td>
<td>0.000</td>
</tr>
<tr>
<td>ABAQUS</td>
<td>ABAQUS/standard</td>
<td>1.000</td>
<td></td>
<td>0.7613</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002015</td>
<td>0.000</td>
</tr>
<tr>
<td>Strand7</td>
<td>Unknown, sparse</td>
<td>0.858</td>
<td></td>
<td>0.7613</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.002015</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Fig. 4.1. Structural model of a 3D building with 6104 beam elements, 4065 nodes, and 24,390 degrees of freedom used to demonstrate the capabilities of the TopFEM software.

Fig. 4.2. Physical model, renderings and topology optimization results of a 3D building with 12,800 brick (B8) elements, 25,760 nodes, and 76,806 degrees of freedom used to demonstrate the capabilities of the TopFEM software (left and central figures courtesy of Skidmore, Owings and Merrill).
dynamic arrays that are automatically managed by the TopS data structure. This grants the analysis application fast indexing of data and avoids memory fragmentation associated with linked lists.

In order to promote the reuse of numerical analysis code, additional helper classes are provided by TopFEM, which can be combined to form specialized element and node attributes. The FemShape class (Fig. 3.2) specifies an element shape function along with all the behavior associated to the interpolation and mapping between natural and Cartesian coordinate spaces. The FemIntegration class (Fig. 3.2) specifies the method and order of numerical integration within an element, and contains the behavior associated to the computation of the integrals over the element. With regards to the different types of analyses that can be conducted, the approach adopted here follows FEMOOP [34] where the object FemElement has an object FemAnalysisModel, which represents its analysis model (see Fig. 3.3). The FemAnalysisModel class is an abstract base class that defines the generic behavior of the different models implemented in the program, such as plane stress, plane strain, and solid.

The FemAnalysis abstract class (Fig. 3.3) defines a common interface for general finite element analyses. It is responsible for forming and solving the governing equations for the finite element model and updating the response quantities at nodes and elements. The main tasks performed in a finite element analysis are abstracted into separate classes as follows: FemIntegrator, FemLinearSolver, and FemSolutionAlgorithm.

The FemIntegrator class is responsible for computing the contributions of the FemElement and FemNode objects to the system of equations. The FemLinearSolver is responsible for storing and solving the systems of equations used in the analysis. The FemSolutionAlgorithm performs an equilibrium analysis on the FemDomain to find the equilibrium state. Examples of subclasses of FemSolutionAlgorithm are FemLinear and FemNewtonRaphson.

In the current implementation we are able to accommodate different constitutive models (e.g. plasticity, hyper-elasticity), large displacement analysis, dynamic loads, among others. In general, the solution of a nonlinear finite element problem requires a time stepping procedure. The approach adopted here follows OpenSees [41] where the FemAnalysis object is a composition of objects from other classes (see Fig. 3.3). To account for nonlinearities, the class FemSolutionAlgorithm, responsible for specifying the steps to solve the equations at the current time step, must find the solution to a nonlinear system of equations (e.g. the Newton–Raphson algorithm). The pseudo-code of such a stepping analysis is illustrated below:

```cpp
for (curr_step = 0; curr_step < m_max_num_steps; curr_step++) {
    m_integrator->newStep();
    m_algorithm->solveCurrentStep();
    m_domain->commit();
}
```

In this pseudo-code, the object m_integrator (e.g. FemNewmark) is responsible for obtaining the matrices and vectors associated to each iteration of the current step; the object m_algorithm (e.g. FemNewtonRaphson) has all the attributes necessary to obtain the solution in the current step; and the object m_linesys (e.g. FemCroutSolver) solves the linear system of equations that arises in each iteration. The pseudo-code below illustrates the Newton–Raphson algorithm.

```cpp
for (curr_itr = 1; curr_itr <= m_max_num_itr && !conv; curr_itr++) {
    m_integrator->computeResidual();
    m_integrator->computeTangent();
    m_linesys->solve();
    m_domain->update();
}
```

In this pseudo-code, the object m_integrator (e.g. FemNewmark) is responsible for obtaining the matrices and vectors associated to each iteration of the current step; the object m_algorithm (e.g. FemNewtonRaphson) has all the attributes necessary to obtain the solution in the current step; and the object m_linesys (e.g. FemCroutSolver) solves the linear system of equations that arises in each iteration. The pseudo-code below illustrates the Newton–Raphson algorithm.

Fig. 4.3. Finite element model and topology optimization results of a 3D building with 24 beam elements, 9120 brick (B8) elements, 18,392 nodes, and 55,176 degrees of freedom.

Fig. 4.4. Time versus number of inserted CZ elements for linear and quadratic tetrahedral meshes [40].

Fig. 4.5. Specimen dimensions (in cm) for the three-point-bending fracture test. The pre-crack is shifted to a position zl, where l is half the distance between the two supports and 0 ≤ z ≤ 1. [40].
In the current framework, the domain is updated after each iteration with the current state of the finite element model. After convergence of the solution algorithm, the current state is committed. Cases where the material models depend not only on the committed/current state have not been considered in the current implementation and further studies are needed to address it.

Furthermore, to include material nonlinearities, subclasses of \texttt{FemMaterial} implement procedures for modifying the stress-strain response to account for different constitutive models. Examples of subclasses of \texttt{FemMaterial} are \texttt{FemMatElasticIso} and \texttt{FemMatVonMises}, as illustrated in Fig. 3.2.

4. Numerical examples

In this section, we analyze and present several examples to highlight the capabilities of the TopFEM program, using both a Crout solver and an UMFPACK (unsymmetric multifrontal sparse LU factorization package) solver [42–45]. These capabilities are demonstrated for a large-scale three-dimensional high rise structure and also in the context of topology optimization of such structures. Some comments are also made on the benefits of extending the TopFEM framework to fracture and fragmentation simulations.

4.1. Finite element analysis and performance

The first example presented here is the analysis of a large three-dimensional structural model of a building. The model includes 6104 beam elements and 4065 nodes, with a total of 24,390 degrees of freedom. The loading and boundary conditions are shown together with the finite element model in Fig. 4.1.

In Table 1, we report the maximum displacements and rotations computed with the TopFEM framework at the top central node of the model (Node 4062) and compare the results to several other structural engineering commercial software packages. We observe that TopFEM with the UMFPACK solver performs comparably to other commercial software packages.

4.2. Topology optimization using the TopFEM framework

An application of the TopFEM framework is evident within the context of topology optimization problems. Topology optimization is a relatively new and powerful tool in the field of structural mechanics, where the goal is to find the optimal layout of a structure within a specified region [46,47]. Using topology optimization, the geometric representation of a structure can be described as similar to a black-and-white rendering of an image with "pixels" given by the finite element discretization [46]. The general optimal design problem can be stated as follows:

\[
\min_{\mathbf{d}} f(\mathbf{d}, \mathbf{u})
\]

\[s.t. \quad g_i(\mathbf{d}, \mathbf{u}) = 0 \quad for \quad i = 1, \ldots, k
\]

\[g_i(\mathbf{d}, \mathbf{u}) \leq 0 \quad for \quad i = k + 1, \ldots, m
\]

where \(f\) is the objective function; \(\mathbf{d}\) is the design field, \(\mathbf{u}\) is the response, which are related through the equality and inequality constraint functions \(g_i\). A canonical example is the minimum compliance problem:

\[
\min_{\mathbf{d}} f(\mathbf{d}, \mathbf{u}) = \mathbf{u}^T K(\mathbf{d}) \mathbf{u}
\]

\[s.t. \quad g_1(\mathbf{d}, \mathbf{u}) = K(\mathbf{d}) \mathbf{u} - \mathbf{p}
\]

\[g_2(\mathbf{d}) = V(\mathbf{d}) - \overline{V}
\]

where \(f\) is the structural compliance, \(g_1\) represents the equilibrium constraint, while \(g_2\) is the constraint on the allowable volume of material, \(\overline{V}\). For demonstrative purposes, we will use the minimum compliance problem throughout this section, though other objectives, such as eigenfrequency, maximum critical buckling load, stress, etc. are also applicable.

For the first problem, topology optimization is performed on the Lotte Tower in Seoul (South Korea) (see [48]). The tower uses an exterior diagrid structure transformed in shape from a square at the base to a circle at the top. The tower is modeled with 12,800 brick (B8) elements and 25,760 nodes, and has 76,806 degrees of freedom.

In Fig. 4.2, the results for the topology optimization for the model are shown, where the analysis at each iteration was performed using the TopFEM framework (see Fig. 4.3).

Next, an example illustrating the combination of discrete and continuum elements for building design (see Fig. 29 of [49]) is studied using topology optimization with the TopFEM framework. Since this framework was written in a generic fashion, the incorporation of different types of elements was straightforward.

4.3. Fracture and fragmentation simulation

Evolution problems such as those involving dynamic fracture and fragmentation can benefit from a topological data structure representation [50–52]. During the fracture process, the numerical simulation requires switching from a continuum to a discrete discontinuity, which can be investigated by means of a cohesive zone model (CZM) [53]. The discontinuity appears as cracks nucleate, initiate and propagate. As indicated by Paulino et al. [40], the topological data structure allows effective insertion of cohesive elements in an arbitrary mesh. Fig. 4.4 shows the elapsed time against the number of cohesive elements for linear and quadratic tetrahedral meshes. Notice that these results show linear scaling with the size of the model, i.e. the time is linearly proportional to the number of inserted cohesive elements. As an example, Fig. 4.5 illustrates a 3D mixed-mode dynamic crack growth problem of a pre-cracked three-point bending beam which was tested by John and Shah [54]. Fig. 4.6 illustrates a typical stress contour and crack pattern for this specimen. Notice that at the time that the simulation is shown (1.5 ms), there are numerous new surfaces that have been created during the damage evolution process. In summary, large-scale simulation of separation phenomena in sol-

\[\text{Fig. 4.6. Stress contour and crack pattern for pre-crack three-point-bending specimen with crack position } x = 0.5 \text{ at } t = 1.5\text{ ms.} \quad [40].\]
ids, such as fracture, branching, and fragmentation, can benefit from a scalable data structure representation, such as TopS, during the evolution process for such moving boundary problems.

5. Conclusions

In this work, we have presented the development of a finite element analysis system, TopFEM, based on the compact topological data structure, TopS [1,2]. This new framework was written to take advantage of the topological data structure together with object-oriented programming concepts to handle a variety of finite element problems, spanning from fracture mechanics to topology optimization, in an efficient, but generic fashion.

The class organization of the TopFEM system was also presented and discussed in comparison to other frameworks in the literature that share similar ideas, such as GetFEM++, deal.II, FEMOOP and OpenSees. The main difference with TopFEM is the way in which the connectivity and topological information for adaptive mesh refinement is handled. The topological data structure is used to represent finite elements of any type, while utilizing the concept of oriented entities to significantly reduce the storage space requirements.

The effectiveness of the TopFEM framework was demonstrated through numerical examples illustrating the analysis capabilities, in addition to its application in topology optimization. We showed how TopFEM can provide a common basis onto which independent finite element analysis codes can be consistently integrated in the topology optimization context. Future implementations in TopFEM will involve evolution problems such as those of Section 4.3.

Acknowledgments

The authors appreciate constructive comments and insightful suggestions from the anonymous reviewers. L.L. Beghini acknowledges support from the National Science Foundation (NSF) Graduate Research Fellowship Program (GFRP). A. Pereira, R. Espinha, I.F.M. Menezes and W. Celes acknowledge the financial support provided by Tcegraf/PUC-Rio (Group of Technology in Computer Graphics), Rio de Janeiro, Brazil. We are thankful to the NSF support through grants CMMI 1321661 and CMMI 13351560. We also appreciate the support from the Donald B. and Elizabeth M. Willett endowment at the University of Illinois at Urbana-Champaign. Any opinion, finding, conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

References

[20] AMD. AMD Core Math Library (ACML); 2011.


