

OBJECT ORIENTED DESIGN PHILOSOPHY FOR SCIENTIFIC COMPUTING

PHILIPPE R.B. DEVLOO¹ AND GUSTAVO C. LONGHIN¹

Abstract. This contribution gives an overview of current research in applying object oriented programming to scientific computing at the computational mechanics laboratory (LABMEC) at the school of civil engineering – UNICAMP. The main goal of applying object oriented programming to scientific computing is to implement increasingly complex algorithms in a structured manner and to hide the complexity behind a simple user interface. The following areas are current topics of research and documented within the paper: hp-adaptive finite elements in one-, two- and three dimensions with the development of automatic refinement strategies, multigrid methods applied to adaptively refined finite element solution spaces and parallel computing.

Mathematics Subject Classification. 65M60, 65G20, 65M55, 65Y99, 65Y05.

Received: 3 December, 2001.

1. INTRODUCTION

Finite element programming has been a craft since the decade of 1960. Ever since, finite element programs are considered complex, especially when compared to their rival numerical approximation technique, the finite difference method. Essential elements of complexity of the finite element method are:

- Meshes are generally unstructured, leading to the necessity of node renumbering schemes and complex sparsity structures of the tangent matrices.
- A separate data structure is required for specifying the boundary conditions.
- Different finite elements are combined into a single code.
- Increasingly complex constitutive equations.

As of the decade of 1990, several other finite element technique need to be incorporated in order to be on the cutting edge of finite element technology:

- h-, p- and/or hp-adaptive finite elements, in order to optimize the finite element mesh;
- multigrid is known to be the most efficient technique to invert the system of equations;
- parallel computing is the most cost effective way to solve large scale problems.

There is no doubt that no single PhD student can be expected to combine all these features into a finite element code and still have the time/energy to make a significant contribution in a field of application. On the other hand, using conventional (*i.e.* structured) programming techniques, it is very difficult for a research laboratory to offer a student an environment which offers these different technologies in a manageable fashion.

Keywords and phrases. Finite element method, object oriented programming, adaptivity, multigrid, substructuring.

¹ Faculdade de Engenharia Civil, UNICAMP, Brazil. e-mail: phil@fec.unicamp.br, longhin@fec.unicamp.br

A second stumbleblock, as seen by the author, is that most research centers which concentrate on finite element research dedicate roughly 90% of the time on teaching applied mathematics and linear algebra and about 10% on programming techniques. On the other hand, students will spend about 10% of their time developing a new idea in finite elements and 90% behind the computer screen writing and debugging their computer code.

The author of this contribution was involved with early development of adaptive finite element technique [7, 8, 12] both in two and three dimensions. It soon became apparent that adaptive techniques could be applied to a variety of problems in computational mechanics and that the main problem was to write an equal number of versions of the adaptive code. Indeed, hundreds of papers have been published different authors showing the applicability of adaptive techniques are effective for any problem [1]. It is unfortunate that most authors had to write their own version of adaptive finite elements.

The original idea of the current project was to write a general purpose finite element program under which most adaptive finite element simulations could be written [11]. Thanks to the contribution of colleagues and students, the scope of the project gradually increased to its current stage.

The program/environment which is described includes hp-adaptive finite elements which can be applied to a variety of simulations, where zero-, one-, two- and tri-dimensional elements of any order of interpolation can be mixed and matched [5]. Elements can be grouped in subdomains to implement sub-structuring algorithms [13]. Different sparse matrix storage formats can be used to store the tangent matrix, and a wide variety of direct or iterative solvers can be used to invert the system of equations. Multigrid acceleration techniques can be applied to nested sequences of meshes. A multi-threaded (multi-)frontal solver allows to solve very large sets of equations with decomposition algorithms [16].

In general, the objectives of the current project could be stated as follows:

- The development of scientific computing tools that can be reused for different problems through:
 - Class structures which implement complex algorithms in a structured manner.
 - Class structures which are as limited in scope as possible. For instance, shape function computations are independent of the mesh structure, matrix objects only implement linear algebra algorithms.
- Promote interoperability between different finite element technologies (adaptivity + multigrid + sub-structuring + multiphysics + nonlinear).
- Extendibility: complexity hidden behind a simplified interface. The refinement of an element is encapsulated in a single call to a method, transferring an element to a substructure item.

This contribution gives an overview of the object oriented environment PZ, in development at Labmec (Laboratório de mecânica computacional) of the school of civil engineering – UNICAMP. The environment is documented using the software Doxygen found at Doxygen <http://www.doxygen.org>, and it can be browsed through the homepage LABMEC <http://labmec.fec.unicamp.br>. The text is not intended to describe the classes and methods in detail. Rather, the authors hope the text will help to comprehend the structure of the environment, which can be browsed at its homepage.

2. COMPARABLE PROJECTS

In this section a list of similar projects is presented. This compilation has no advertisement purpose, it is a result of a one to two level web search on the current topic¹.

One can easily note that most of those works started in academic institutions backed by high technology industries such as aerospace, oil exploration, defense department and so on. Some of the projects have spawned off commercial packages while others still maintain their academic roots.

¹The keyword searched was Object Oriented Finite Elements. Then we looked for links within all the pages listed. We selected those projects with active links and with sufficient web based documentation and which are related to our work.

OFELI library

OFELI stands for Object Finite Element Library, as described in OFELI <http://www.lma.univ-bpclermont.fr/touzani/ofeli.html>, (no citations about OO programming available) is an object oriented library of C++ classes for development of finite element codes. According to the information contained on the web site, the package is fully documented. The following features are noted:

- Implements the most common matrices storage schemes such as full, skyline, sparse and tri-diagonal for both symmetric and non-symmetric morphologies.
- Implements shape functions and finite elements in same classes but differs from the authors proposal in a sense that different interpolation degree define different classes.
- The solvers are not implemented as objects, such treatment results in a less generic implementation. From the end-users perspective (one implementing a application with your library) the linear algebra complexities are still present. In a case where solvers are implemented as objects, most of the parameters involved are treated by the solver class, hides complexity.
- Does not implement adaptivity.
- Different from the authors proposal, geometrical and computational meshes are the same.
- It is a multi-physics package, implements Heat Transfer, Solid Mechanics, Laplace, Fluid Dynamics and Electromagnetics equations.
- A generic material class implements the desired functionality of such entities. This approach is very similar to ours.
- It is a multi-platform package, runs under Windows and Unix ware operational systems.
- The package redefines a few BLAS functions. In terms of usability this is a good proposal but as a result some performance loss could be noted.
- It is distributed under GNU license.

PHLEX Kernel

According to information obtained in PHLEX <http://www.comco.com/phlex/phlex.html>, [15] this package is a kernel that implements hp-adaptive techniques as well as flexible object-based database, a selection of linear and nonlinear solvers, error estimation, automatic mesh refinement, a GUI and post processing functionalities. Since PHLEX is a commercial package, the necessary information for a more profound analysis of the package is not available. On its web pages the following features are highlighted.

- Implements error estimation and its functionalities.
- Implements adaptivity in a sense that this feature becomes an error control functionality.
- Is structured so that it contemplates vector/parallel architectures.
- Implements different kinds of solvers (unspecified) for a variety of problems addressed by PHLEX.
- Its data structure is based on an object-oriented paradigm, and contains Objects and Methods, as well as an Object-independent library for manipulating the Objects.

Deal

According to information obtained in DEAL <http://gaia.iwr.uni-heidelberg.de/deal/> and [2,6] this package is an object oriented C++ program library targeted at adaptive finite elements and error estimation. It addresses in a efficient manner all the complexities involved in adaptive finite element codes. The following features are noted.

- Support for one, two, and three space dimensions, using a unified interface that allows to write programs almost dimension independent. It seems to be quite similar to the authors proposal.
- Handling of locally refined grids, including different adaptive refinement strategies based on local error indicators and error estimators.
- Support for a variety of finite elements, including Lagrange elements of order one through four, and discontinuous elements.

- The documentation is indeed extensive, but at first glance it seems a bit confusing.
- Support for several output formats, including some common formats for visualization of scientific data. Very useful feature, since an independent data visualizer can be used. The authors package also implements such features, directing its output format to former IBMs DataExplorer, now known as OpenDX.
- Support for a variety of computer platforms.
- On multiprocessor machines, many operations are parallelized.
- Free source code under an Open Source license, and the invitation to contribute to the further development of the library.

VectorSpace vs.lib and fe.lib

The information available in VectorSpace <http://www.vector-space.com/> (no specific references) describes two C++ libraries with emphasis on object oriented finite element methods. In a more generic perspective addresses applications in advanced numerical analysis such as computational linear algebra, linear programming, unconstrained/constrained optimization, finite difference method, boundary element method, and variational methods.

The more generic vs.lib implements general algebra functionality, on the other hand fe.lib implements the more specific finite elements functionalities. The available information advises that, in order to get the best out of those two libraries, using them together is recommended.

Since those two libraries are commercial ones, documentations and informations are a bit restricted. A deeper investigation of that package would require a more open documentation. At first glance the package seems to be quite interesting.

3. THE IMPACT OF OBJECT ORIENTED PROGRAMMING ON SCIENTIFIC COMPUTING

It would be elusive to imagine the package presented was written in its current form from the start. The program started as a non-adaptive code which allowed the integration of different simulations. Then, p-enrichment was added. H-refinement was written initially using the concepts developed in earlier work on h-adaptivity. This refinement procedure needed to be totally rewritten in order to accommodate tri-dimensional hp refinements. Finally substructuring was incorporated implementing a substructure as a double derivation of a mesh and a computational element.

At each extension of the program, the core classes needed to be adapted. The adaptations, however, were mainly in the line of turning the interface of the classes increasingly consistent.

Object oriented software development has been fundamental in the organization of the project. The strong separation of the boundaries of the classes permitted to coordinate the efforts of several researchers on a single project. Most of the maintenance of the program is to simplify its structure.

The strong separation between class boundaries and attributes facilitates the documentation of the environment. The main difficulty of the user is to understand the concept which lays behind the definition of each class. It is even more difficult to explain how these concepts interact between each other to obtain the desired numerical simulation.

4. ALGEBRAIC CLASSES

Most scientific algorithms can be viewed as a sequence of matrix-matrix and matrix-vector operations. Therefore, an object oriented environment for scientific computing needs to be based on algebraic matrix classes [18]. Algebraic matrix classes must provide the user with:

- A simple interface to create matrix objects.
- Allow efficient manipulation of the data in the matrix objects.
- Have a strong mathematical/numerical underlying concept to facilitate maintenance.

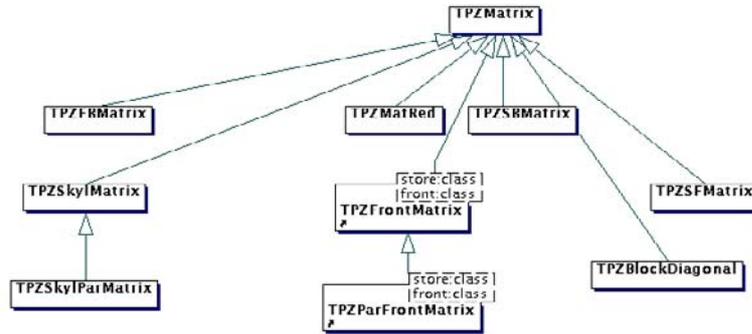


FIGURE 1. Class tree for matrices.

The algebraic classes used do not implement expression templates, nor do they try to optimize multiplication or addition operators. Where possible algebraic operations are executed by calls to the BLAS or ATLAS library package.

4.1. Matrix classes

Even though many object oriented packages exist for matrix computations, no packages known to us attended our needs. The interface needed presents us with an abstract matrix class from which an extensive family of matrix storage patterns are derived. The underlying concept of the matrix class is a linear transformation from R^n to R^m . The minimum behavior of any derived matrix class is to multiply the matrix by a matrix of compatible dimension and put the result in a third matrix. Element selection (*i.e.* a_{ij}) need not be implemented.

The matrix class and its derived classes implement the operator() method as *non virtual*. Instead, the operator() method of the base matrix class calls a s(int i, int j) method which is virtual. This implies that, if the class type is known, non virtual element access can be performed through the simple operator() interface.

At the abstract level, the matrix class implements the decomposition methods LU, LDLt and Cholesky and provides an interface to the iterative methods implemented by the *Iterative templates routines* available on netlib [3].

The most important derived matrix class is TPZFMATRIX, which implements the full matrix storage form. The TPZFMATRIX class is particular as it is storage declared (*i.e.* its elements are stored column wise for FORTRAN compatibility). This implies that the elements of the TPZFMATRIX objects can be accessed by pointer arithmetic in those stretches of code where execution time is critical. This is a point where safety of access is traded for efficiency.

The many storage patterns for classes are typical for finite element computations. In what follows some storage patterns are described, together with their use (see Fig. 1):

- Skyline storage form: (TPZSkyline) combines sparse storage with speed of decomposition.
- Full storage form: (TPZFMATRIX) used for all purposes, also for column vectors.
- Banded storage form: (TPZSBndMatrix) efficient storage pattern for one-dimensional problems.
- Block banded storage form: (TPZBlockDiag) mainly used as a preconditioning matrix for iterative methods.
- Overlapping block storage form: (TPZOvIblock) this class implements the element by element storage form and/or can be used as a preconditioner for iterative methods, where the support of the blocks to be inverted is larger than the block diagonal.
- Frontal matrix storage form: (TPZFrontMatrix) this class is derived from the root matrix class, but has the peculiarity that it decomposes the equations and stores them on secondary storage as they are

being assembled. No pre-front operation is necessary as the front size is increased dynamically during the computations.

4.2. Solution procedures

Efficient finite element computations often depend on complex algebraic solution procedures. Whereas two dimensional problems can generally be solved with direct solvers (*e.g.* frontal decomposition), tri dimensional problems usually generate systems of equations too large to be decomposed directly.

The class structure which implements solution procedures provides the user with a structured environment to develop iterative methods.

The base class is `TPZSolver`, which defines the interface to the methods *Solve* and *Clone*. The *Solve* method applies a solution process to the right hand side and corrects the current solution. The *Clone* method creates a copy of the current object. The main class derived from `TPZSolver` is `TPZMatrixSolver`, which implements a solution process associated with a matrix object. `TPZMatrixSolver` implements a variety of solvers such as:

- direct methods: LU, LDLt and Cholesky;
- iterative methods: Jacobi, SOR, SSOR;
- krylov space methods: Preconditioned Conjugate Gradient and GMRES.

The preconditioner applied to the krylov space methods is another instance of the `TPZSolver` class. This allows to formally implement any type of preconditioner to the CG and GMRES methods.

`TPZMatrixSolver` has a pointer to a `TPZMatrix` object plus a reference counter. When a second `TPZMatrixSolver` object references the same matrix, the reference counter is increased. The matrix object is deleted only when its reference counter is equal zero.

The `TPZMatrixSolver` is a simple interface to the methods which are available in the base matrix class. It is an object which, through the *Solve* method presents an interface to the variety of solution methods implemented in the base matrix class.

Two other noteworthy derivations of `TPZSolver` are `TPZStepSolver` and `TPZMGSolver`.

`TPZStepSolver` implements a solution procedure as a sequence of solution procedures. This can be a sequence of subdomain inversions, each with a different color. It can be a smoothing step, followed by a multi-grid step followed by a smoothing step.

`TPZMGSolver` is constructed based on a transfer matrix and a `TPZSolver` object. It multiplies the right hand side by the transfer matrix, applies the solve method of the `TPZSolver` object, and finally applies the transpose of the transfer matrix to obtain the corrected solution.

The idea of associating a solution method with an object has given an additional flexibility to the scientific computing environment.

5. OBJECT ORIENTED FINITE ELEMENTS

Object oriented finite element development seized to be an innovative topic some years ago. The notion of associating a finite mesh, a finite element and a node with an object is quite natural [14, 17, 19]. The current project goes beyond the translation of traditional finite element structures to create an environment that allows to aggregate many finite element technologies in a single scope without compromising cpu time and/or memory usage. In order to make the environment accessible as few classes as possible were created, multiple derivations were avoided wherever possible and template classes were used only where their benefit is substantial.

In order to optimize memory usage and increase execution speed, dynamic memory is avoided wherever possible. Nodes are allocated in chunks (thousand at a time), and elements do not allocate any dynamic memory even though they implement arbitrary orders of interpolation. The code is structured in such a way that an element stiffness matrix can be computed without a single dynamic memory allocation. This attitude was taken after a profile operation on a previous version of the code indicated that 90% of the CPU time was spent in dynamically allocating and releasing memory space.

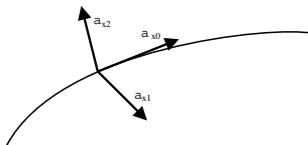


FIGURE 2. Axes associated with a line element.

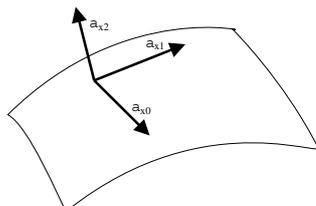


FIGURE 3. Axes associated with a surface element.

The class structure which underlies the finite element approximation is based upon the mathematical description of the finite element method, as taught at the Texas Institute of Computational Mechanics TICAM <http://www.ticam.utexas.edu>:

- First the domain of the differential equation is approximated by geometric finite elements. The geometry of these elements is defined by the mapping of a *master element* to a deformed element. This mapping can be obtained by the Lagrangian interpolation functions, but other mappings are possible.
- On each geometric element, an interpolation space is defined. The order of interpolation over each element can be arbitrary, but in order to ensure continuity, the order of interpolation over sides shared between two elements has to be identical and the coefficients associated with these shape-functions have to coincide. This interpolation space will be used to approximate the solution of the differential equation².
- The variational statement associated with the differential operator allows to associate a residual vector with an element of the interpolation space. The finite element solution is the element of the interpolation space which zeroes the residual vector. The tangent matrix to the residual vector is called the global stiffness matrix. The residual vector (and global stiffness matrix) is obtained as the integration of the variational statement over the element. The contributions of the variational statement at the integration points is implemented in a separate class structure.

5.1. Geometric approximation

Within the PZ environment, the geometric classes are responsible for the approximation of the domain of the differential equation and for keeping track of the topology of the mesh.

The domain of the differential equation is approximated by a set of geometric nodes and elements. The geometric element is obtained as the injective map between a master element and the tri-dimensional Euclidean space. A one-dimensional element is a curve in space, a two-dimensional element is a hyperplane in space. A point element is a point in space.

The Jacobian of the mapping between the master element and deformed element corresponds to the partial derivatives of the (ax_0, ax_1, ax_2) coordinates with respect to the parameter space [10]. (ax_0, ax_1, ax_2) correspond to a local orthogonal system associated with the deformed element and can be different at each integration point (see Figs. 2 and 3).

²Note that the interpolation space is independent of the differential equation.

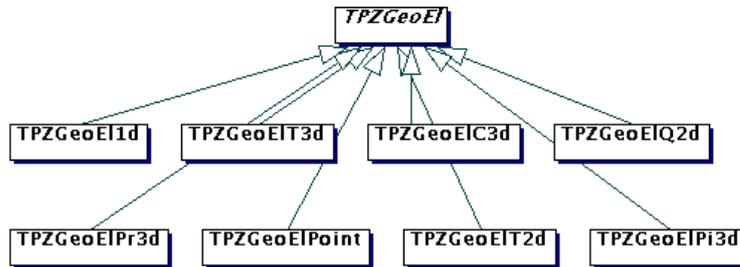


FIGURE 4. Implemented element topologies.

The directions of (ax_0, ax_1, ax_2) are provided by the call of the Jacobian method³. The complete set of geometric elements implemented are: point element, line element, quadrilateral, triangular, tetrahedra, prism, pyramid and hexahedra (see Fig. 4). The mapping from the master element to the deformed element is performed by Lagrangian interpolation functions.

An experiment has been performed in which the interpolation is computed in a cylindrical, spherical or toroidal coordinate system. As a result a linear element mapped perfectly on a spiral, a quadrilateral element formed part of a cylinder, sphere or torus. Other mapping functions can be implemented where necessary, only a simple class derivation is needed.

5.2. Topological refinement of a mesh

A second attribute of the geometric elements is to keep track of the topology of the mesh. Under topology, it is understood the element node/connectivities and element/neighbor information. The element/node connectivity corresponds to the traditional finite element datastructure. The neighbouring information extends the traditional notion of neighbours and is described as follows [4]:

Each geometric element is a closed set of points and is formed by the union of its sides. Each side is an open set of points. As such:

- a linear element has three side: two sides of dimension zero and one side corresponding to the interior of the element;
- a triangular element has seven sides: three sides of dimension zero (the corner nodes), three sides of dimension one (the border of the element) and one side of dimension two (the interior nodes of the triangle);
- a quadrilateral element has nine sides;
- a hexahedral element has twenty seven sides;
- etc.

Each side of an element can be formed by a bijective map of a master element of the corresponding topology. Two elements are neighbours along a side if the sides correspond to identical sets of points. This implies that two elements can be neighbours along more than one side.

The datastructure used to keep track of the element/node connectivity and neighbouring information is:

- geometric elements contain a vector of geometric nodes;
- geometric elements contain a vector of pointers to neighbouring elements/sides⁴.

³For a point element, the three axes are arbitrary, for a one dimensional element, the second axes needs to be specified externally.

⁴This datastructure is quite large, especially for three dimensional element (*e.g.* an overhead of 54 words for each hexahedral element). On the other hand, this information can be reconstructed based on the element/node connectivity information. It is a trade-off between execution speed and storage.

The relationship between sub-elements and their father is also based on side information. A father element/side is the father of a subelement/side iff the father element/side contains all points of the subelement/side. To each subelement/side corresponds exactly one father element/side. This implies that there exists a transformation from the parametric space of the subelement/side to the parametric space of the father element/side in which it is contained.

The refinement of a geometric element consists in the creation of subelements contained in the father element and updating the topology information of its neighbours.

5.3. Definition of the interpolation space

The definition of the interpolation space is implemented by a separate class structure. To each computational element corresponds a geometric element. The computational element contains a pointer to the geometric element. The geometric element is used to compute the Jacobian of the mapping between the master element and the deformed element and also to report the topology of the mesh.

For each element topology (*i.e.* linear, quadrilateral, triangular) a set of hierarchical shape functions is defined⁵, which are compatible between each other. This implies that a one-dimensional element can be neighbor of tetrahedral element which is neighbor of a pyramid, retaining the continuity of the interpolation space.

The shape functions for each element are divided in sets of shape-functions associated with the sides of the corresponding geometric element. A support of a shape function associated with a side contains only that side and the sides of higher dimension. The number of shape functions associated with a side depends on the order of interpolation along that side. The number of sets of shape functions is equal to the number of the sides of the corresponding geometric element.

To each set of shape functions a connectivity object is associated (class *TPZConnect*). The connectivity object stores the global equation number associated with the shape functions of a side. In order for the global interpolation space to be continuous, the following conditions must be met:

- If two computational elements are neighbor along a side, then they must share the *TPZConnect* object.
- The number of shape functions associated with the *TPZConnect* object must be the same for all neighbouring elements (*i.e.* the order of interpolation for the side must be the same).

Contrary to the geometric mesh, which contains the complete tree of refined elements, the computational elements form a partition of the domain. In fact, several computational meshes can be built from a single geometric mesh. This one to many relationship between the geometric mesh and the computational mesh forms the basis of the implementation of the multigrid method.

A computational mesh which contains elements of different level is an adaptive mesh. In an adaptive mesh, there are elements which share points along a side but which are not neighbours. In these cases the shape functions associated with a side which is contained in another side need to be constrained with respect to all shape functions associated with the closure of the larger side. This concept is commonly known as *hanging nodes*. In the framework of PZ, a better name would be *hanging sides*, because shape functions are not necessarily associated with nodes.

Restrains are computed by performing an L2 projection of the side shape functions of the smaller element to the side shape functions of the larger element. The implementation of the projection is made possible by the knowledge of the mapping between the smaller element/side to the larger element/side provided by the corresponding geometric elements.

The coefficients of the constraints are stored within the corresponding *TPZConnect* object.

⁵In order to improve reusability, the shape functions implementation are located in a separate directory. In this sense, any researcher who wants to experiment with hierarchical shape functions can download this directory, without carrying the complete finite element package.

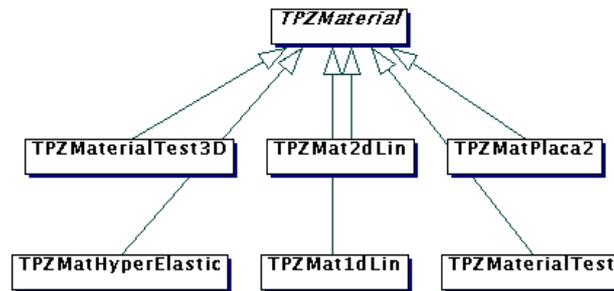


FIGURE 5. Class tree structure for different variational statements.

5.4. Abstraction of the variational statement

The third step in the approximation of the finite element method is to use the interpolation space to construct an approximate solution of the differential equation and/or its equivalent variational statement [9].

It is assumed that a residual can be computed based on the integration of test functions multiplied by a differential operator which depends on the location of the integration point, the function value and its derivative and, eventually, the size of the element (*e.g.* SUPG type formulations). The computation of the contribution to the residual at an integration point is implemented in the classes derived from *TPZMaterial*. As a concept the classes derived from *TPZMaterial* have the following attributions:

- Compute the contribution to the residual vector and tangent matrix at an integration point, based on the values of the shapefunctions, their gradient, the value of the solution, its gradient, the weight of the integration point and the direction of the axes of the Jacobian.
- Compute the contribution of the boundary conditions at an integration point.
- Compute post processed variables, depending on the physical problem being modeled. Post processing by the material object is computed based on a method which computes the post processed variable based on the location of the point, the value of the solution and its gradient.

This setting is general enough to allow implementation of a large variety of approximations of physical problems (see Fig. 5):

- one- two and tri-dimensional scalar problems (*e.g.* heat transfer problems);
- one-dimensional curved beams;
- one-dimensional conservation laws (Linear transport, Burgers law, Euler equation);
- two-dimensional elasticity;
- curved plates and shells coupled with beams;
- two-dimensional conservation laws (Burgers equation, Euler equation);
- tri-dimensional hyper-elasticity;
- two-dimensional flow through porous media.

Other simulation can readily be added. The classes which define the variational statement are independent of the definition of the interpolation space and/or the finite element mesh. This implies that the development of the variational statement can be done independently of the finite element package as a whole.

5.5. Adaptive refinement of the interpolation space

As mentioned earlier, the PZ environment consists in two separate mesh definitions: the definition of the geometry and the definition of the interpolation space. The geometric mesh provides the information of the topology of the mesh and contains the tree of the refinement structure. The interpolation space corresponds to a partition of the computational domain. Restraints on adaptivity shape function coefficients are computed during element creation and destruction. Within this framework h and hp- adaptivity is virtually built in:

- Refinement consists in the destruction of a computational element followed by the construction of a set of refined elements.
- Coarsening consists in the destruction of a group of elements, followed by the construction of a larger element.

The method which computes the restraints is implemented in a method of the base class, independent of the dimension of the side associated with the restraint. The derived element classes implement the methods which are necessary to compute the values of the shape functions along the sides.

5.6. Sub-structuring of the mesh

A sub-structure (class *TPZSubCMesh*) is a mesh which contains only part of the computational domain. As such a sub-structure has a father mesh, which differentiates it from the root mesh. A sub-structure has a double behaviour: On the one hand it is a mesh because it contains a collection of computational elements and *TPZConnect* nodes, on the other hand it is an element because it transfers a residual vector and tangent matrix to its father mesh.

Sub-structuring is implemented as a double derivation from a computational element and a computational mesh. It makes a distinction between internal connects and external connects. The external connects are the equations which will be contributed to the father mesh, whereas the internal connects are the equations which will be condensed locally.

The main difference between the assembly of the sub-mesh and the assembly of a *regular* mesh is that, after assembly, equations associated with internal nodes need to be condensed onto the interface nodes.

An analysis procedure is associated with the *TPZSubCMesh* class, which allows the user to choose the matrix storage pattern and solution procedure (direct symmetric/non symmetric or iterative solver).

Within the framework of sub-structuring, sub-structures can be embedded in other sub-structures, without restriction to the number of levels.

A frontal matrix solution procedure has also been implemented. The frontal decomposition procedure is naturally associated with the static condensation of equations. A frontal solution procedure where several element matrices are computed in parallel, when combined with sub-structuring naturally leads to a multi-frontal solution procedure.

5.7. Multigrid operators

Several interpolation spaces can be derived from a single geometric mesh. Therefore, two elements from distinct interpolation spaces can relate to each other through their association with a single geometric mesh.

A method is implemented at the level of the computational mesh which computes a transfer matrix which maps a solution on the coarse mesh to a solution on the fine mesh. The fine mesh needs to be embedded in the coarse mesh, otherwise the interpolation will not be exact. In order to initiate a multigrid procedure the following steps need to be performed:

- Create a coarse mesh.
- Create a solution procedure for the coarse mesh (*e.g.* a direct solver).
- Create a fine mesh.
- Compute the transfer operator between the coarse and fine mesh.

The V-cycle multigrid method is implemented as a solution procedure whose solution sequence is (*TPZStep-Solver*)

- Smoothing step.
- Transfer of the residual to the coarse mesh.
- Solution procedure on the coarse mesh.
- Transfer of the solution correction of the coarse mesh to the fine mesh.
- Smoothing step.

The three steps: *Transfer residual to coarse mesh, solve coarse mesh, transfer solution correction to fine mesh* are implemented in a derived class *TPZMGSolver*. Additional levels of the multigrid procedure can be added by refining the last mesh and using multigrid procedure as solution procedure for the last but one mesh.

5.8. Embedded refinement

Embedded refinement is a refinement technique where a small mesh is superposed on top of a regular mesh. Therefore, instead of refining the element mesh to zoom in on the singularity, a small element mesh is superposed on the coarse mesh. This technique is useful for simulations where a small scale phenomenon is embedded in large domain computations. As examples one can cite:

- A reentrant corner in an elasticity computation often represents a singularity:
 - A crack embedded in an elastic domain.
 - A boundary layer computation within an Euler grid.

The modeling of these phenomena using adaptive elements and/or sophisticated meshing procedures present the inconvenience that adaptive meshes or mesh generators will generally smoothly zoom the mesh towards the singularity. This could be considered as a waste of computer resources.

Embedded meshes allow one to superpose a fine mesh on an existing coarse mesh. The shape functions associated with the fine mesh are summed to the shape functions of a regular mesh to form a refined interpolation space. The idea, which would be very difficult to implement in a structured framework (based on data structures and functions), naturally implements within the object oriented PZ framework.

A single template class *TPZGlobalLocal* is defined which implements the interpolation space and computation of residual and stiffness matrix respectively. This class implements the concept of embedded spaces for the complete family of finite elements defined in the PZ environment.

The embedded elements support the hp adaptive element spaces and computation of transfer matrices between embedded meshes. Embedded elements can naturally be sub-structured, having as main effect the modification of stiffness matrix and right hand side of the element within which the small elements are embedded.

6. STRUCTURAL MATRICES

Global stiffness matrices and preconditioner matrices (*e.g.* blockdiagonal, overlapping block, etc.) need to extract their information from the finite element mesh in a particular way, dependent on their internal data structure. This implies that, either the computational mesh or the matrix class needs to be extended to provide the information or extract the information from the other object. Extending these classes in such a way would either couple the matrix classes to the finite element classes or *vice-versa*. The solution presented is to create a class whose responsibility is to build the structure of the matrix class based on the information of the mesh. As a result a strong separation is obtained between the definition of algebraic classes and finite element classes.

Structural matrices are classes associated with algebraic matrix classes whose responsibility is to create assembled matrices (and right hand sides) with appropriate computation procedures. Structural matrices have a significant impact on the ease with which an analysis procedure can change the type of matrix to work with. Before working with these coupling classes, the code of the analysis procedures needed be modified each time a different global matrix type was used. For instance:

- Skyline matrix needs to know the skyline of the matrix at construction.
- Banded matrix needs to know the bandwidth of the matrix.
- Frontal matrix needs to reorder the elements before assembly.
- Block diagonal matrix needs to integrate only the block diagonal.

7. SOLUTION PROCEDURES

Thus far, the different components were described which compose a finite element computation: a computational mesh (which includes the definition of the variational statement), the algebraic matrix classes, solution procedures and structural matrix classes. The interaction between these components is grouped in the *TPZ-Analysis* class. Thus far, there are only three supported analysis types: Linear analysis, which is the base class, Nonlinear analysis, which iterates the solution procedure until convergence and the sub-structuring analysis, which implements the static condensation procedure.

8. POST PROCESSING OF FINITE ELEMENT SOLUTIONS

Graphical post processing is necessary when computing with hp adaptive meshes. The numbers generated by hierarchical shape functions make no physical sense. The approach taken for graphical post-processing is to develop a class structure which generates output for a graphical post processor. The reason for creating yet another class structure is that for p-refined meshes, the elements need to output more data than the data associated with their corner nodes. The graphical elements generate a logical refinement internally computing the positions and indices of the internal nodes without actually creating the elements.

The only graphical package used currently is OpenDX, because it is extendible and programmable and also because it is distributed as public domain software. Other graphics packages have been supported in the past.

9. CURRENT DEVELOPMENTS

1. Implement arbitrary refinement patterns: the hp adaptivity is being entirely rewritten. The next version will allow arbitrary refinement patterns in any dimension, as long as the Jacobian of the sub-elements is constant.
2. Accelerate multi-grid iteration for hp-refined meshes. The multigrid method for hp-adaptive methods is operational and has been tested on different problems. The convergence rates obtained are very slow however, due to inadequate smoothing preconditioners. We expect that the overlapping block preconditioner with or without mesh coloring will yield an effective multi-grid method.
3. Implement auto-adaptive hp-refinement strategies. Preliminary work has been performed with Dr Demkowicz during a one month stay in Texas to develop an adaptive strategy which automatically decides upon the use of h-, p- or hp-refinement. This work is being extended by Edimar Cesar Rylo during his master thesis.
4. Develop an object oriented environment for grid computing. Having stabilized the adaptive hp refinement strategies, the natural following step is to adequate the environment to parallel computing. The line of work is to develop a class structure to interface the pool of computers available. Some preliminary has been done by Erico Correia da Silva who implemented the conjugate gradient algorithm on a IBM/SP2 machine using the environment OOPAR.
5. Performance assessment of the environment. The main focus of the work has been in the development of adequate class structures to implement advanced finite element algorithms. Care has been taken that all operations are local in scope (*e.g.* the refinement of an element is a local operation, involving only the neighbouring elements). Numerical experiments demonstrate that more than 80% of the CPU time is spent in the decomposition of the global stiffness matrix. Nevertheless, a critical assessment of the numerical efficiency of all components needs to be done.
6. Graphical user interface to perform numerical experiments. The PZ environment offers a wide variety of possible numerical simulations. The challenge is to bring these possibilities to the end user in a as simple framework as possible. Two developments are considered:
 - (a) write a graphical user interface using Qt to allow the user to choose the global matrix type, solution method, etc.;
 - (b) write a Python interface for the main classes of the environment, offering the more knowledged user a interpreted environment to develop algorithms with PZ.

10. CONCLUSIONS

Without object oriented technology, it would not have been possible to develop a finite element code incorporating all presented features. Therefore, the question is not whether C++ is as efficient as FORTRAN, but what can be achieved with C++ as compared to FORTRAN. The next challenge is how to put this complexity to work in order to obtain significant results.

Most of the features presented have a relatively simple interface. Applying a refinement is a call to a single method from an element object, transfer an element to a sub-structure is a call to a method of the sub-structure object.

Careful programming has allowed to cross-check different finite element technologies: adaptivity, sub-structuring, multi-grid, variety of differential equations in three dimensions. Many combinations have not been verified, but the class structure should allow them to be used.

A major benefit of using an object oriented structure is the possibility to generate understandable documentation automatically. We especially acknowledge the public domain tool Doxygen and the company Object International TogetherSoft <http://www.o.i.com> which makes free licenses available to their software Together.

Acknowledgements. This work has been realized in collaboration with colleagues and students. All have both benefited from and contributed to the PZ environment. The financial support of FAPESP, CAPES, CNPq and of the companies Petrobras and Commodity is also gratefully acknowledged.

REFERENCES

- [1] Ivo Babuska Ahmed K. Noor, Quality assessment and control of finite element solutions. *Finite Eleme. Anal. Des.* **3** (1987) 1–26.
- [2] W. Bangerth, Using modern features of C++ for adaptive finite element methods: Dimension-independent programming in deal.II, in *Proceedings of the 16th IMACS World Congress 2000, Lausanne, Switzerland, 2000*, M. Deville and R. Owens Eds. (2000). Document Sessions/118-1.
- [3] R. Barrett, M. Berry, T.F. Chan, F. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM (1994).
- [4] C.M.A.A. Bravo, *Sobre a implementação da técnica hp-adaptativa tri-dimensional para elementos finitos*. Ph.D. thesis, Faculdade de Engenharia Mecânica, UNICAMP (2000).
- [5] C.M.A.A. Bravo, P.R.B. Devloo and R. Pavanello, Sobre a implementação do refinamento h-p in *CILAMCE99 Congresso Ibero Latino Americano de Metodos Computacionais para Engenharia* (1999) 1–21.
- [6] B. Cockburn, G. Kanschat, I. Perugia and D. Schötzau, Superconvergence of the local discontinuous galerkin method for elliptic problems on cartesian grids. Technical Report 2000/71 (2000).
- [7] L. Demkowicz, P.R.B. Devloo and J.T. Oden, On an h-type mesh refinement strategy based on minimization of interpolation errors. *Comput. Methods Appl. Mech. Engrg.* **1-2** (1986) 63–87.
- [8] P.R.B. Devloo, A three-dimensional adaptive finite element strategy. *Comput. Structures* **38** (1991) 121–130.
- [9] P.R.B. Devloo, On the development of a finite element program based on the object oriented programming philosophy, in *Proc. First Annual Object-Oriented Numerics Conference*, Corvallis, OR, USA (1993) 183–203. Object Oriented Numerics Conference, Rogue Wave Software, Inc.
- [10] P.R.B. Devloo, PZ: An object oriented environment for scientific programming. *Comput. Methods Appl. Mech. Engrg.* **150** (1997) 133–153.
- [11] P.R.B. Devloo and J.S.R.F. Alves, On the development of a finite element program based on the object oriented programming philosophy, in *Numer. Methods Engrg. '92*, Ch. Hirsch, O.C. Zienkiewicz and E. Oñate Eds., Elsevier, Brussels, Belgium (1992) 39–42. First European Conference on Numerical Methods in Engineering, Elsevier.
- [12] P.R.B. Devloo, J.T. Oden and P. Pattani, An hp-adaptive finite element method for the numerical simulation of compressible flow. *Comput. Methods Appl. Mech. Engrg.* **70** (1988) 203–235.
- [13] P.R.B. Devloo and E.C. Rylo, Implementação da técnica de subestruturação no ambiente de elementos finitos pz. *XXII Iberian Latin-American Congress on Computational Methods in Engineering* (2001).
- [14] B.W.R. Forde, R.O. Foschi and S.F. Stierner, Object-oriented finite element analysis. *Comput. Structures* **34** (1990) 355–374.

- [15] T.J. Liszka, C.A.M. Duarte and O.N. Hamzeh, *Hp-meshes method for dynamic fracture propagation*. The computational Mechanics Company, Inc. (COMCO) (1999).
- [16] G.C. Longhin and Ph.R.B. Devloo, An object oriented multi-threaded multi-frontal solver. *XXII Iberian Latin-American Congress on Computational Methods in Engineering* (2001).
- [17] R.I. Mackie, Object oriented programming of the finite element method. *Internat. J. Numer. Methods Engrg.* **35** (1992) 425–436.
- [18] M.M.L. Santana and P.R.B. Devloo, Object oriented matrix classes. *AIMETA Joint conference of Italian group of Comput. mech. CILAMCE* (1996) 325–328.
- [19] T. Zimmermann and Y. Dubois-Pelerin, The object-oriented approach to finite elements: concepts and implementations. *Proc. of the First European Conference on Numer. Methods Engrg.*, Brussels (1992) 865–870.