Eidgenössische Technische Hochschule Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

The Application of Object Oriented Methods to Boundary Elements

C. Lage

Research Report No. 96-19
October 1996

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

The Application of Object Oriented Methods to Boundary Elements[1]

C. Lage [2]

Seminar für Angewandte Mathematik
Eidgenössische Technische Hochschule
CH-8092 Zürich
Switzerland

Research Report No. 96-19 October 1996

Abstract

In this paper we present the design of a class library to support the development of software for boundary elements. We discuss the extensibility and reusability of the library and give an example of its application.

Keywords: object oriented design, boundary element methods

[1] submitted to Elsevier Science

[2] supported by the German Research Foundation, Ha 1324/6-5

1 Introduction

Finite element methods (FEM) and boundary element methods (BEM) are the most important discretization techniques when dealing with partial differential equations or integral equations, respectively. Numerous software packages implementing these methods exist each more or less tailored to suit specific problems or requirements. But all of them are based on the same concepts, e.g. the *test* and *trial spaces*, *bilinear forms* or the *finite element* itself, which is called a *panel* in the boundary element context. Each of these generic concepts is certainly implemented in a slightly different way considering the various software packages but always with the same functionality. To improve the software development, i.e. mainly to reduce the development time and to increase the reliability, the generic concepts should be provided by a library or framework to ensure their reusability.

In this paper we present the design and application of such a framework for boundary elements using object-oriented methods. But the same approach is valid for finite element methods. We chose an object-oriented approach for several reasons. Most importantly, it supports an extensible design of the library necessary to increase its applicability. For example, the developer of specific software could introduce new problem-oriented quadrature rules that again can be used by higher level operations of the library, like matrix generation, without having access to the sources of the library. Moreover, even advanced methods for the compression of the fully populated system matrix of boundary element methods, could be added sharing the overall architecture of the library. For the case of the panel clustering method, which imposes high demands on the administration of panels, this is shown in Section 4. But the same statement remains valid for further methods like wavelet algorithms. A consequence of this kind of extensibility is that applications based on the library could easily incorporate new techniques, which were not available when implementing the library. For example, switching from standard boundary element applications with dense matrices to applications using the panel clustering formulation does not need much effort.

This flexibility is not provided by traditional programming techniques like structured programming and cannot be realized using their associated programming languages, e.g. Pascal, C or FORTRAN. For the implementation of our library we use the language C++[7]. It is widely available, efficient and fits the numerical needs.

In object-oriented methods the modelling of a software system is based on real world concepts. They are used to decompose the system into small cooperating units, called objects. These objects are described by classes that are related in a hierarchy. This kind of ranking is used to express common parts of objects. With an abstract base class for example we are able to specify the outside view or interface of an object. Subclasses of the base class, so-called concrete classes, are used to realize the behaviour assigned by the interface.

This construction is applied several times in our classification, for example, in the specification of basis functions discussed in Section 3.2. For a comprehensive discussion of object-oriented methods it is worthwhile to study the book of Grady Booch [1].
In the case of boundary element methods the 'real world' concepts correspond to the concepts of the mathematical formulation, i.e. the formulation of Petrov-Galerkin schemes or projection methods. Moreover the formulation enables us to skip the analysis part in the software development cycle such that we can proceed with the object-oriented design directly.
The paper is organized as follows. In the next Section we give a short review of the notation of boundary element methods. Subsequently we sketch the design of the class library (Section 3) and discuss the embedding of the panel clustering method (Section 4). Finally in Section 5 the application of the library is illustrated with a simple example.

2 Preliminaries

Let $\Omega \subset R^d$ be a bounded domain with a piecewise analytic boundary manifold $\Gamma := \partial\Omega$ and let V be a Hilbert space, e.g. $L^2(\Gamma)$ or $C(\Gamma)$. We consider the boundary integral equation on Γ

$$Au = f \tag{1}$$

for the unknown density $u \in V$ with

$$(Au)(x) := \int_\Gamma k(x,y)\, u(y)\, dy. \tag{2}$$

The integral equation is usually derived by a reformulation of a boundary value problem in Ω via the integral equation method. For the numerical solution of (1) we focus on the weak formulation:

For given right-hand side $f \in V$ find $u \in V$ such that

$$\langle v,\, Au\rangle = \langle v,\, f\rangle \qquad \forall v \in V'. \tag{3}$$

Here V' denotes the dual space of V and $\langle\cdot,\cdot\rangle$ the dual form on $V' \times V$. Replacing V and V' by finite dimensional subspaces $V_n := \mathrm{span}\{\psi_i\}_{i<n}$ and $V_n' := \mathrm{span}\{\phi_i\}_{i<n}$, $n \in N$, yields the Petrov-Galerkin method to obtain an approximate solution $u_n \in V_n$. Finally, inserting the basis functions leads to the desired system of linear equations:

$$A_n u_n = f_n \tag{4}$$

with $A_n := (\langle \phi_i,\, A\psi_j \rangle)_{i,j<n}$ and $f_n := (\langle \phi_i,\, f \rangle)_{i<n}$.

Example 1 *Choosing Dirac delta functions δ_{ξ_i} concentrated at suitable points $\xi_i \in \Gamma$ as test functions ϕ_i characterizes the collocation method, thus the coefficients of the system matrix A_n are determined by evaluating*

$$\langle \phi_i,\, A\psi_j \rangle = \int_\Gamma k(\xi_i,\, y)\psi_j(y)\, dy. \tag{5}$$

Identifying V and V', one obtains the Galerkin method. In this case the dual form is identical to the scalar product in V:

$$\langle \phi_i,\, A\psi_j \rangle = \int_\Gamma \phi_i(x) \int_\Gamma k(x,\, y)\, \psi_j(y)\, dy\; dx. \tag{6}$$

3 The Class Library

3.1 Geometry

For the representation of the geometry, i.e. the boundary Γ, we propose two stages of description. The first, the *physical model*, denotes a partition of the boundary Γ in elementary parts Γ_j, $0 \le j < J$, so-called *patches*:

$$\Gamma = \bigcup_{j<J} \Gamma_j \tag{7}$$

which are, for example, specified by charts κ_j and can carry problem-oriented attributes. In the second stage, the *numerical model*, an approximation of the boundary by means of geometric objects, such as (curved) triangles or (curved) quadrilaterals, which can be handled efficiently by numerical algorithms, e.g. the quadrature rules, is constructed. The instantiation of these geometric objects, called *panels*, uses the information supplied by the physical model as indicated in the class diagram [1] in Figure 1, where the physical and the numerical model are denoted by the classes *Boundary* and *Panelization*, respectively. With the instantiation of several approximations according to different levels we are able to construct a hierarchy of approximations necessary to realize multilevel methods or wavelet methods.
Besides the flexible generation of approximations, the representation of the geometrical information in two stages serves to specify an interface to the numerical treatment of the problem. To say, changes in the preprocessing of the

[1] We use the notation of Booch as described in [1]

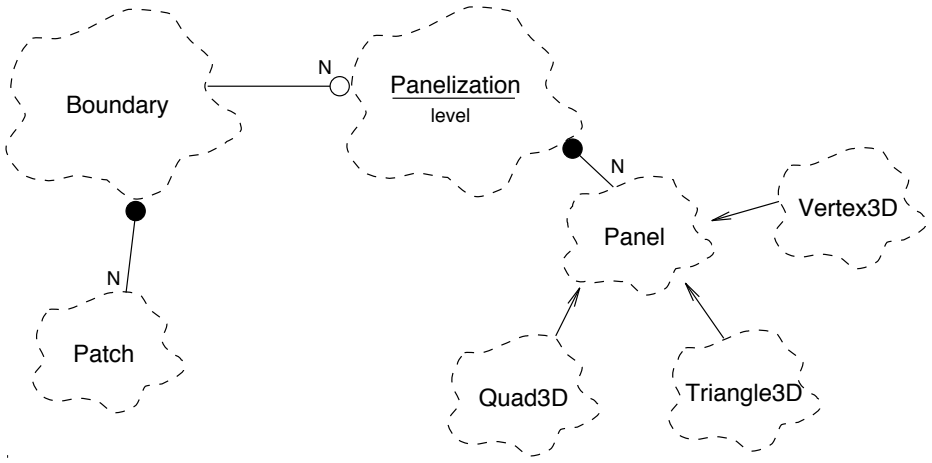

Fig. 1. Geometry architecture

geometry are handled by specializations of the class *Boundary*, whereas the class *Panelization* and all other numerical components are not affected.

3.2 Subspaces

The Petrov-Galerkin discretization is defined by the choice of the subspaces V and V'. In the design they are characterized by a collection of objects representing basis functions which we have to address now.
Basis functions are expressed indirectly via their non-vanishing parts on each panel, known as shape functions. For every panel the linear combination of all corresponding shape functions forms a panel function. For the abstraction of panel functions a hierarchy of abstract base class and concrete classes, i.e. an interface with distinct implementations, as mentioned in the Introduction, is used. The abstract base class *PanelFunction* captures the structure and behaviour that are common for different types of panel functions by means of pure virtual functions:

```
class PanelFunction {
  public:
    virtual const Panel&  support() const = 0;
    virtual unsigned      dimension() const = 0;
    virtual int           index(unsigned i) const = 0;
};
```

As illustrated by the class definition above, these are

– the information about the underlying panel,
– the number of shape functions associated with the panel,
– the index of the basis functions corresponding to the shape functions.

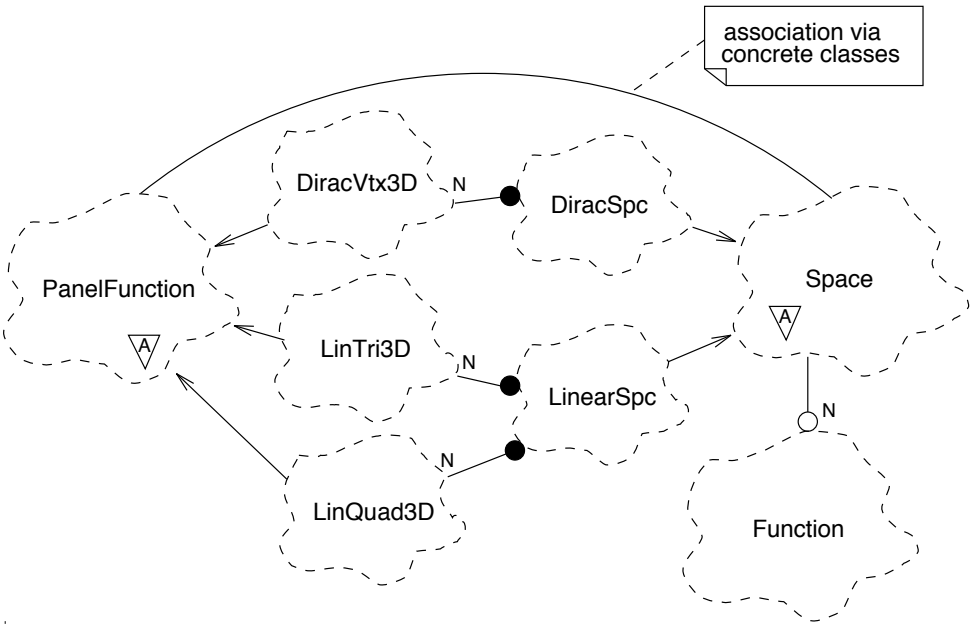

Fig. 2. Subspace architecture

An example for the implementation of a panel function is the class definition of the concrete class *LinTri3D*, an abstraction used to model piecewise linear functions on triangles:

```
class LinTri3D : public PanelFunction {
    const Triangle3D*  supp_;
    int                idx_[3];

  public:
    const Panel&   support() const { return *supp_; }
    unsigned       dimension() const { return 3; }
    int            index(unsigned i) const { return idx_[i]; }
};
```

The instantiation of panel functions is provided by the class *Space*, again an abstract base class. Implementations of *Space* realize various subspaces used in the Petrov-Galerkin discretization, e.g. piecewise linear functions or Dirac delta functions (Figure 2). Global constrains, for example, conforming indices to model continuous functions, can be handled during the instantiation process.
Arbitrary functions of the subspaces are indentified with the vector of coefficients according to the chosen basis. In the library, this vector is wrapped in the class *Function*, that, in addition, supplies elementary operations like addition, scaling or inner product.

In the design, the essential characteristic of integral operators is the mapping of functions, or, in our terms, the mapping of objects of the class *Function*. We objectify this mapping in an abstract base class as follows:

```
class Operator {
  public:
    virtual void operator()(const Function& f, Function& g) = 0;[2]
};
```

In the case of standard boundary element methods, i.e. when no compression of the system matrix is applied, the usage of a two-dimensional array to implement the operator is obvious. This is covered by the class *MatrixOp*:

```
class MatrixOp : public Operator {
    real* matrix_;
  public:
    MatrixOp(const Space& test, const Space& trial, const DualForm& df);
    void operator()(const Function& f, Function& g);
    ⋮
};
```

The constructor of the class is responsible to evaluate the system matrix A_n depending on the test space, the trial space and the dual form specified in the parameter list. With the class *DualForm* an abstraction of the type of integral operator by means of suitable quadrature rules is given. This topic is discussed more accurately in [3]. The pure virtual function defined in the abstract base class is now overloaded with a standard matrix-vector product.
With this design we render *Operator* objects as active objects. The advantage of this design decision is, that we can easily change the algorithms computing the mapping of an operator using derived classes (Figure 3). This is for example necessary for reasons of efficiency as in the case of the mass matrix. The sparsity of this matrix makes the use of an array obsolete. More adequate, for instance, are hash tables, thus an implementation of the matrix-vector product consistent with this kind of data structure is required (class *SparseMatOp*). A similar reason for the flexibility of the evaluation of matrix-vector products is given by the application of compression techniques as discussed in the next section.
In contrast to the previous examples, where new classes are derived to com-

[2] If we supply an analysis of expression trees to avoid the generation of temporary objects by delayed evaluation, this member function could be declared to support a more natural notation, namely g = A(f) instead of A(f, g) as proposed here.

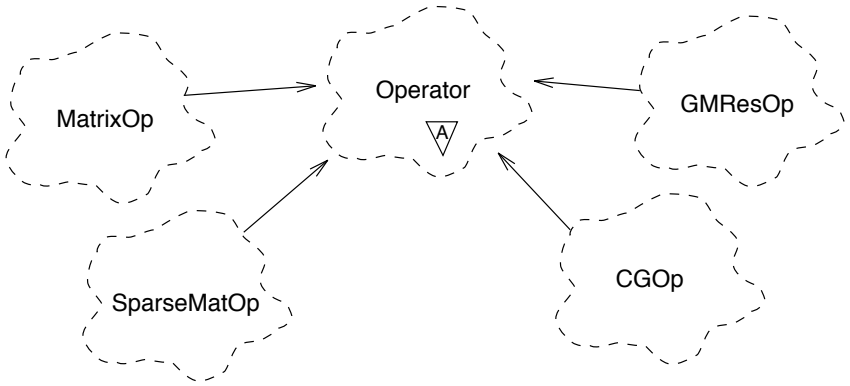

Fig. 3. Operator architecture

pensate structural variations, the following extends the application of the abstraction. Solving the discrete system can be read as applying the inverse of the considered operator. Hence, a solver is a kind of operator, thus we derive it from the base class *Operator* and overload the mapping with the solution process (classes *CGOp* and *GMResOp*). In particular, these classes can be initialized with every object that is an *Operator*, i.e. that is derived from base class *Operator*, because they only rely on the mapping of the operator to be inverted.

4 Embedding of the Panel Clustering Method

Due to the nature of integral operators stemming from the reformulation of boundary value problems, the boundary element method leads to full system matrices. Strictly speaking, this property is caused by the strong coupling of x- and y-dependent terms if kernel functions of convolution type occur. To suppress the coupling by means of factorization, the panel clustering method developed by Hackbusch and Nowak (cf. [2,6,4]) replaces the kernel with an expansion. [3] This yields an approximation of the system matrix which can be written in terms of sparse (rectangular) matrices:

$$A_n \sim A_n^{\mathrm{pc}} = A_n^{\mathrm{near}} + A_n^{\mathrm{far}}\, U_n. \tag{8}$$

Hence, time and storage requirements for the discretization of boundary integral equations are substantially reduced.
To preserve the sparseness the product of the rectangular matrices A_n^{far} and U_n cannot be evaluated in advance. Therefore, each matrix-vector product $A_n^{\mathrm{pc}} u_n$

[3] Similar ideas, known as fast multipole methods, are used in the case of Nyström discretizations [5].

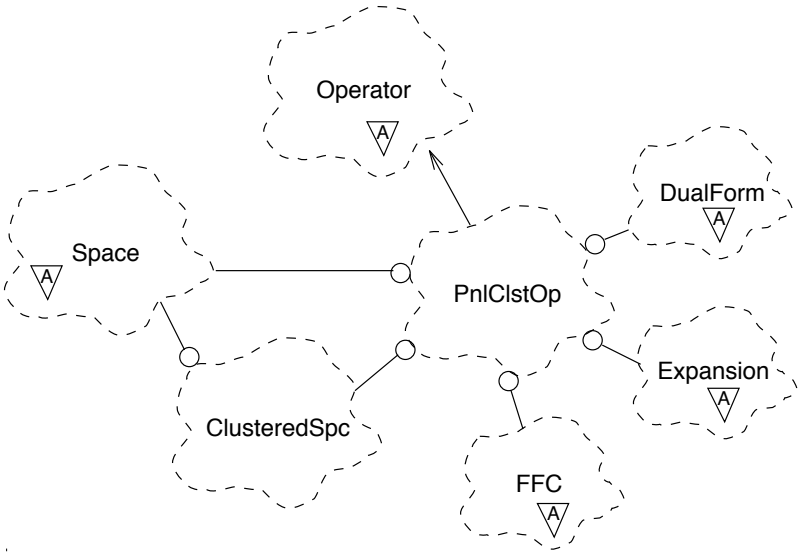

Fig. 4. Panel clustering architecture

must be replaced with the operation

$$A_n^{\text{near}} u_n + A_n^{\text{far}} \left(U_n u_n \right). \tag{9}$$

For a detailed discussion of the panel clustering method we refer to [4].
The panel clustering algorithm is another candidate for an implementation of the abstract base class *Operator*, i.e. we derive a further class *PnlClstOp* inheriting the behaviour of *Operator* (Figure 4). This ensures the employment of panel clustering in all applications that rely only on that interface, e.g. solvers as explained in Section 3.3.
As in the case of the *MatrixOp* the setup of the three matrices representing the panel clustering approximation is organized by the constructor of *PnlClstOp*. The necessary information about the considered integral operator are provided by the classes *DualForm*, *Expansion* and *FFC*. Furthermore, the panel clustering imposes access to set of panels, so-called clusters, that are related in a hierarchy. The access is prepared and handled by objects of the class *ClusteredSpc*.

5 An Example

As an example for the application of the class library we consider the following boundary value problem:

Let $\Omega \subset R^3$ be the unit cube. For given $f \in L^2(\Omega)$ find $u^{\text{bv}} \in H^1(\Omega)$ such that

$$\Delta u^{\text{bv}} = 0 \qquad \text{in } \Omega \tag{10}$$

```
int main() {
  Boundary      bnd("cube");
  Panelization  pnl(bnd, 2);

  LinearSpc     trlspc(pnl);
  DiracSpc      tstspc(pnl);

  LaplaceDlp    dlp(2, 2, 0.25);
  Mass          mass;

  MatrixOp        K(tstspc, trlspc, dlp);
  SparseMatOp     M(tstspc, trlspc, mass);

  LiCoOp        A(-0.5, M, 1.0, K);
  GMResOp       invA(A, 1e-8, 100);

  Function f(tstspc);
  Function u(trlspc);

  f = "(sin(pi*x*1.1) * cos(pi*y*1.3) * cosh(sqrt(2.9)*pi*z))";
  invA(f, u);
}
```

Lst. 1. Implementation of the problem using the class library.

$$u^{\mathrm{bv}} = f \qquad \text{on } \Gamma := \partial\Omega \tag{11}$$

We seek the solution u^{bv} of the Dirichlet problem in form of a double-layer potential with unknown density u on Γ:

$$u^{\mathrm{bv}} = -\frac{1}{4\pi} \int_{\Gamma} \frac{\langle n_y,\, y - x \rangle}{\|y - x\|^3}\, u(y)\, dy, \qquad x \in \Omega. \tag{12}$$

This approach leads to the boundary integral equation

$$Au = \left(-\tfrac{1}{2} I + K\right) = f \quad \text{on } \Gamma, \tag{13}$$

where

$$(Ku)(x) := -\frac{1}{4\pi} \int_{\Gamma} \frac{\langle n_y,\, y - x \rangle}{\|y - x\|^3}\, u(y)\, dy. \tag{14}$$

The discretization of equation (13) by Petrov-Galerkin methods is specified when trial and test spaces are chosen (Section 2). Listing 1 shows the implementation of the collocation method using the class library. Due to the close relation of the mathematical concepts the explanation of the listing is straightforward. After the generation of the physical and numerical informations of the geometry represented by the objects *bnd* and *pnl*, respectively, the trial and test space are chosen (*trlspc*, *tstspc*). Here, we apply piecewise linear

Table 1
Collocation method, piecewise linear functions.

# of panels	# of vertices	CPU-time [s]	real time [s]	e
192	98	0.6	0.4	4.19e-01
768	386	4.8	2.9	1.12e-01
3072	1538	61.5	37.2	2.85e-02
12288	6146	966.0	583.8	7.20e-03

functions and Dirac delta functions concentrated in the vertices according to collocation methods. In the next two lines the dual form implied by the double-layer potential (cf. (5)) as well as the dual form associated with the identity are declared and defined. In addition, parameters concerning the quadrature rules, e.g. the number of Gaussian nodes, are given. In the example the integral operator is described by a linear combination of the identity and the double-layer potential. Again, this representation is carried over into the implementation by defining an object of the class *LiCoOp*. This class is another implementation of the base class *Operator* and is responsible for evaluating the linear combination of two *Operators*, in our example the linear combination of the mass matrix *M*, i.e. the discrete identity, and the discrete double-layer potential *K*. Finally, the discrete integral operator *A* together with further parameters, e.g. a stopping criterion, is used to initialize the solver denoted in the listing by the operator *invA*. In the last line *invA* is applied to the given right hand side *f* to solve the boundary integral equation (13).
We run the program of Listing 1 on a *SUN Ultra-Enterprise* with two *UltraSPARC* processors and 512Mb of RAM. For several panelizations the results are listed in Table 1. The CPU-time measured is the time to compute (assembling + solving) an approximation of the density u. Because we are using a multiprocessor machine, i.e. the task is divided between the processors, this time exceeds the real time.
Inserting the discrete density in (12) yields an approximation u_n^{bv} of the solution u^{bv}, which is known in advance, if we use harmonic functions for the right hand side f (cf. Listing 1). Therefore, we can easily evaluate the error in internal points of the domain. The error listed in Table 1 is the average of relative errors in various points:

$$e := \frac{\sum_{x\in P} |u_n^{\mathrm{bv}} - u^{\mathrm{bv}}|}{|P|}, \quad \text{with} \tag{15}$$
$$P := \left\{ 0.2\,(i,\, j,\, k)^T \in R^3 : \, i,j,k \in \{1,2,3,4\} \right\}.$$

Instead of using the Collocation method we could discretize (13) by means of the Galerkin method. We switch between these methods by changing the declaration and definition of *tstspc* in Listing 1 to

```
LinearSpc    tstspc = trlspc;
```

Table 2
Galerkin method, piecewise linear functions.

# of panels	# of vertices	CPU-time [s]	real time [s]	e
192	98	1.1	0.7	4.81e-01
768	386	12.2	7.4	1.24e-01
3072	1538	173.6	104.5	3.23e-02
12288	6146	2807.5	1687.0	8.16e-03

Table 3
Galerkin method, piecewise constant functions.

# of panels	# of vertices	CPU-time [s]	real time [s]	e
192	98	1.0	0.7	5.11e-01
768	386	11.9	7.1	1.42e-01
3072	1538	169.7	102.8	3.62e-02

Table 2 lists the results; whereas Table 3 refers to the case of employing piecewise constant functions, i.e.

```
ConstSpc   trlspc(pnl);
ConstSpc   tstspc = trlspc;
```

So far, the double-layer potential is represented by a full matrix. As discussed in Section 4 we can apply the panel clustering method to construct an approximation of the matrix with substantially reduced time and storage consumptions. To employ the method in our example, we only have to replace the *MatrixOp* by a *PnlClstOp* and provide the environment for the latter to operate:

```
ClusteredSpc      clsttrlspc(trlspc);

MonomialFFC       ffc(degree);
DMonomialFFC      dffc(degree);
LaplaceExp        exp(degree);

PnlClstOp  K(tstspc, clsttrlspc, dlp, ffc, dffc, exp, eta, degree);
```

The classes *MonomialFFC*, *DMonomialFFC* and *LaplaceExp* are implementations of the abstract base class *FFC* and *Expansion*, respectively. No further changes in Listing 1 are necessary. Table 4 shows the calculation for a panelization that would need 4.6GB to store the full matrix. With the panel clustering method we were able to compute the solution on our machine using only 445MB of storage and about 3% of the computing time compared to the standard method.

Table 4
Galerkin method, piecewise linear functions, panel clustering.

# of panels	# of vertices	CPU-time [s]	real time [s]	e
49152	24578	1033.4	639.5	4.89e-02

References

[1] Grady Booch. *Object-oriented Analysis and Design*. Benjamin / Cummings, Redwood City, California, second edition, 1994.

[2] W. Hackbusch and Z.P. Nowak. On the Fast Matrix Multiplication in the Boundary Element Method by Panel Clustering. *Numerische Mathematik*, 54(4):463–491, 1989.

[3] Christian Lage. Object-oriented design aspects for boundary element methods. In W. Hackbusch and G. Wittum, editors, *Boundary Elements: Implementation and Analysis of Advanced Algorithms*, volume 54 of *Notes on Numerical Fluid Mechanics*, Braunschweig, Germany, 1996. Vieweg Verlag.

[4] Christian Lage. *Softwareentwicklung zur Randelementmethode: Analyse und Entwurf effizienter Techniken*. PhD thesis, Universität Kiel, 1996.

[5] V. Rokhlin. Rapid solutions of integral equations of classical potential theory. *J. Comput. Phys.*, 60:187–207, 1985.

[6] Stefan Sauter. *Über die effiziente Verwendung des Galerkinverfahrens zur Lösung Fredholmscher Integralgleichungen*. PhD thesis, Christian-Albrechts-Universität Kiel, 1992.

[7] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, second edition, 1991.

Research Reports

No.	Authors	Title
96-19	C. Lage	The Application of Object Oriented Methods to Boundary Elements
96-18	R. Sperb	An alternative to Ewald sums. Part I: Identities for sums
96-17	M.D. Buhmann, Ch.A. Micchelli, A. Ron	Asymptotically Optimal Approximation and Numerical Solutions of Differential Equations
96-16	M.D. Buhmann, R. Fletcher	M.J.D. Powell's work in univariate and multivariate approximation theory and his contribution to optimization
96-15	W. Gautschi, J. Waldvogel	Contour Plots of Analytic Functions
96-14	R. Resch, F. Stenger, J. Waldvogel	Functional Equations Related to the Iteration of Functions
96-13	H. Forrer	Second Order Accurate Boundary Treatment for Cartesian Grid Methods
96-12	K. Gerdes, C. Schwab	Hierarchic models of Helmholtz problems on thin domains
96-11	K. Gerdes	The conjugated vs. the unconjugated infinite element method for the Helmholtz equation in exterior domains
96-10	J. Waldvogel	Symplectic Integrators for Hill's Lunar Problem
96-09	A.-T. Morel, M. Fey, J. Maurer	Multidimensional High Order Method of Transport for the Shallow Water Equations
96-08	A.-T. Morel	Multidimensional Scheme for the Shallow Water Equations
96-07	M. Feistauer, C. Schwab	On coupled problems for viscous flow in exterior domains
96-06	J.M. Melenk	A note on robust exponential convergence of finite element methods for problems with boundary layers
96-05	R. Bodenmann, H.J. Schroll	Higher order discretisation of initial-boundary value problems for mixed systems
96-04	H. Forrer	Boundary Treatment for a Cartesian Grid Method
96-03	S. Hyvönen	Convergence of the Arnoldi Process when applied to the Picard-Lindelöf Iteration Operator
96-02	S.A. Sauter, C. Schwab	Quadrature for *hp*-Galerkin BEM in $\mathbb{R}^3$
96-01	J.M. Melenk, I. Babuška	The Partition of Unity Finite Element Method: Basic Theory and Applications