SoftwareX 11 (2020) 100476

Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

Bembel: The fast isogeometric boundary element C++ library for Laplace, Helmholtz, and electric wave equation

J. Dölz [a], H. Harbrecht [b], S. Kurz [c], M. Multerer [d], S. Schöps [c], F. Wolf [c,*]

[a] TU Darmstadt, Department of Mathematics, Germany
[b] Universität Basel, Department of Mathematics and Computer Science, Switzerland
[c] TU Darmstadt, Institute TEMF & Centre for Computational Engineering, Germany
[d] Università della Svizzera italiana, Institute of Computational Science, Switzerland

ARTICLE INFO

Article history:
Received 3 June 2019
Received in revised form 28 February 2020
Accepted 2 April 2020

Keywords:
BEM
IGA
Laplace
Helmholtz
Maxwell
C++
FMM
$\mathcal{H}^2$-matrix

ABSTRACT

In this article, we present Bembel, the C++ library featuring higher order isogeometric Galerkin boundary element methods for Laplace, Helmholtz, and Maxwell problems. Bembel is compatible with geometries from the Octave NURBS package, and provides an interface to the Eigen template library for linear algebra operations. For computational efficiency, it applies an embedded fast multipole method tailored to the isogeometric analysis framework and a parallel matrix assembly based on OpenMP.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_176
Legal Code License	GPL3
Code versioning system used	Git
Software code languages, tools, and services used	C++, OpenMP
Compilation requirements, operating environments & dependencies	C++11, Eigen Linear Algebra Library
If available Link to developer documentation or manual	http://temf.github.io/bembel/
Support email for questions	info@bembel.eu

1. Introduction

The *boundary element method* (BEM) or *method of moments* (MoM) is a widely used tool for the solution of partial differential equations (PDEs) in engineering applications such as acoustic and electromagnetic scattering problems in homogeneous media. It is accepted in industry and implementations are available as software packages. Prominent examples are BEM++ [1] which is open-source and BETL [2] which is freely available for academic use. A commonality of these codes is that they are relying on mesh generators to create triangulation-based surface meshes consisting of flat triangles or lower order parametric elements. Thus, the order of convergence of (higher-order) boundary element methods is limited by the approximation error of the underlying mesh.

With the emergence of *isogeometric analysis*, see [3], boundary element methods have received increased attention by the community, since *computer aided design* (CAD) tools directly provide parametric representations of surfaces in terms of *non-uniform rational B-splines* (NURBS). This has recently led to a flourishing development of software concerning NURBS, see, e.g., [4,5].

* Corresponding author.
E-mail addresses: doelz@mathematik.tu-darmstadt.de (J. Dölz), helmut.harbrecht@unibas.ch (H. Harbrecht), kurz@gsc.tu-darmstadt.de (S. Kurz), michael.multerer@usi.ch (M. Multerer), schoeps@temf.tu-darmstadt.de (S. Schöps), wolf@temf.tu-darmstadt.de (F. Wolf).

https://doi.org/10.1016/j.softx.2020.100476

Since the extraction of volume mappings from surface descriptions is an active research area with open problems, the use of isogeometric finite element methods is challenging in practice. Boundary element methods are the methods of choice in this setting, see also [6–12]. However, their implementation usually requires a significant amount of expert knowledge, which can lead non-experts to refrain from their usage.

The software library Bembel, **B**oundary **E**lement **M**ethod **B**ased **E**ngineering **L**ibrary, is a header-only library written in C++ (compliant to the 2011 standard) [13]. It provides a general framework and building blocks to solve boundary value problems governed by boundary integral operators within the isogeometric framework, for example the Laplace, Helmholtz, or electric wave equation. The development of the software started in the context of *wavelet Galerkin methods* on parametric surfaces, see [14], where the quadrature routines for the Green's function of the Laplacian have been developed and implemented. It was then extended to *hierarchical matrices* ($\mathcal{H}$-matrices) in [15] and to *$\mathcal{H}^2$-matrices* and higher-order B-splines in [16]. With support of B-splines and NURBS for the geometry mappings, the Laplace and Helmholtz code became isogeometric in [7]. In [17], it has been extended to be applicable to non-scalar problems, by covering the electric field integral equation, and was modernized to a flexible C++ header-only library in early 2020.

The publication of this software package aims at making isogeometric boundary element methods available for a broader audience. Therefore, we aim at an easy to use C++ API, which streamlines the access to the underlying routines, with compatibility to the Eigen template library for linear algebra, see [18]. This is achieved, while still providing black-box $\mathcal{H}^2$-compression of the boundary element system matrices and OpenMP parallelization. The $\mathcal{H}^2$-compression yields an almost linear computational cost in the number of unknowns for assembly, storage requirements and matrix–vector multiplication of the system matrix. For the representation of geometries, Bembel features arbitrary parametric mappings, most prominently given as NURBS-mappings, which can directly be imported from files generated by the Octave NURBS package [4].

The structure of this document is as follows: Section 2 gives a brief introduction to isogeometric boundary elements, referring to the Appendix for details. Then, Section 3 explains our design considerations, after which Section 4 showcases a specific code example. With some familiarity to the important classes of our API, Section 5 and Section 6 then give an explanation of the structure of our library and its major building blocks. This is followed by a showcase of some simple numerical results in Section 7, obtained in complete analogy to the first code example. Afterwards, some remarks regarding the impact of our implementation are given in Section 8, and, finally, we conclude in Section 9.

2. Isogeometric boundary element methods

Bembel is able to treat general boundary value problems based on boundary integral operators, and provides examples to solve the well-known Laplace, Helmholtz, and electric wave equations as stated in Appendix A out of the box. For the Helmholtz and the electric wave equation, the provided examples assume non-resonant wave-numbers.

The boundary value problems can be solved by *single layer potential ansatzes*, see Appendix B, which require the solution of *boundary integral equations* by means of a numerical discretization. This and similar approaches are commonly known as *boundary element methods*. Bembel implements these through the use of a conforming Galerkin scheme.

One of the inherent advantages of boundary element methods over classical finite element techniques in the volume is that they can act directly on surface descriptions by NURBS from CAD programs, see Appendix C. This, in connection with corresponding spline spaces, leads to so-called *isogeometric boundary element methods*. Bembel implements these and assumes that the surface descriptions fulfil the requirements stated in Appendix D. The spline spaces for the Galerkin method are constructed as isogeometric multi-patch B-spline spaces, see Appendix E.

It is well known that in all provided examples solvability and uniqueness of the solution, both of the continuous problem as well as of its discrete counterpart, can be guaranteed by imposing reasonable assumptions on the boundary values, essentially guaranteeing "physicality" of the input data, see e.g. [19,20].

The computation of the system matrix requires the evaluation of singular integrals, see [20]. For the numerical quadrature of these integrals, we employ regularization techniques as described in [21]. The compression of the resulting densely populated system matrices is based on the embedded fast multipole method (FMM), which is tailored to the framework of isogeometric analysis, see [16,17], and fits into the framework of $\mathcal{H}^2$-matrices. Its particular advantage is that the matrix compression is directly applied on the reference geometry, that is, the unit square. Hence, the employed compression scheme profits from the inherently two-dimensional structure of the problem. The cost for assembly, storage requirement and matrix–vector multiplication for the system matrix are almost linear in the number of unknowns, see [16,17]. Moreover, this compression technique provably maintains the convergence behaviour for increasingly finer discretizations, cf. [16]. An in-depth mathematical analysis of the implemented approach together with numerical studies based on previous versions of Bembel is available in [7,16,17].

3. Design considerations

Most modern three-dimensional boundary element codes with built-in matrix compression are written in C or C++, resulting in efficient implementations. One of the central aims of Bembel is to provide a computationally efficient isogeometric boundary element code with a plain and simple user interface mimicking the mathematical setting. Therefore, the API of Bembel is designed in modern, template-based C++ and provides an interface for the Eigen template library for numerical linear algebra. This allows the user for a programming experience similar to Matlab and Octave and for the use of all matrix-free algorithms from the Eigen library. Particularly, all solvers provided by Eigen can be employed without further modifications.

4. Example program

Following the example of the Eigen library, Bembel is designed as a header-only library, structured into different modules. Before we discuss these modules in detail, we will present an example program to familiarize the reader with the typical program flow.

The discussed example is a main file that solves a Laplace problem and evaluates the potential at user defined points in space. The code corresponds to one of the examples provided in Bembel's repository, shortened by omitting loops over the polynomial degree and the refinement level, error measurements, and console output.

All of the presented functionality in the following example is either in the Eigen or Bembel namespace, thus lines 2 and 3 are merely for convenience. Line 4 initializes a Geometry object from a given file. As mentioned before, we support geometry files as exported by the NURBS package of Octave, and provide examples in the official repository that showcase how users can generate their own.

```
1  int main() {
2    using namespace Bembel;
3    using namespace Eigen;
4    Geometry geometry("external_geometry_file.
       dat");
5
6    MatrixXd gridpoints = // user defined points
7
8    std::function<double(Vector3d)> fun =
9      [](Vector3d in) {  // user defined right
       hand side
10     };
11
12   int polynomial_degree = // runtime parameter
13   int refinement_level = // runtime parameter
14
15   AnsatzSpace<LaplaceSingleLayerOperator>
16     ansatz_space(geometry, refinement_level,
       polynomial_degree);
17
18   DiscreteLinearForm<DirichletTrace<double>,
19                      LaplaceSingleLayerOperator
       >
20     disc_lf(ansatz_space);
21   disc_lf.get_linear_form().set_function(fun);
22   disc_lf.compute();
23
24   DiscreteOperator<H2Matrix<double>,
       LaplaceSingleLayerOperator>
25     disc_op(ansatz_space);
26   disc_op.compute();
27
28   ConjugateGradient<H2Matrix<double>, Lower |
       Upper, IdentityPreconditioner> cg;
29   cg.compute(disc_op.get_discrete_operator());
30   auto rho = cg.solve(disc_lf.
       get_discrete_linear_form());
31
32   DiscretePotential<
       LaplaceSingleLayerPotential<
       LaplaceSingleLayerOperator>,
33                     LaplaceSingleLayerOperator
       > disc_pot(ansatz_space);
34   disc_pot.set_cauchy_data(rho);
35   auto pot = disc_pot.evaluate(gridpoints);
36
37   return 0;
38 }
```

Lines 8–10 define a function in $\mathbb{R}^3$ that is used to generate boundary values, in the form of a `std::function` for convenience. Afterwards, in lines 12 and 13 the polynomial degree of the basis functions and the refinement level are set. Both are runtime parameters, thus Bembel allows for h, p and k refinement.

In line 15–16, the discrete space is initialized. The `AnsatzSpace` class generates the correct ansatz space for the `LaplaceSingleLayerOperator`, or possibly another linear operator defined by the user. The constructor of the `AnsatzSpace` class, offers a default parameter for knot repetition, which can be used for the refinement as well, thus making it possible to differentiate between p and k refinement.

Lines 18–22 assemble the right hand side of the linear system. The `LinearForm`, in this case given by the `DirichletTrace` implements a mapping that evaluates `fun` on the parametric surfaces w.r.t. the reference domain, which has to be passed to the `LinearForm` in line 21. Line 22 then calls the appropriate quadrature routines and assembles the right hand side.

Lines 24–26 take care of the assembly of the system matrix. In this example, we choose to assemble the system as our own and Eigen compatible `H2Matrix` format, but `Eigen::MatrixXd` may be used as well for a non-compressed assembly. The `AnsatzSpace` object is passed to the `DiscreteOperator` in line 25.

Therein, the `compute()` method then assembles the matrix. The specialization of the `LinearOperatorBase` class given via the `LaplaceSingleLayerOperator` needs to provide an `evaluate_integrant_impl` implementation, which is used to compute the matrix entries. A possible implementation is discussed below.

In lines 28–30, the linear system is solved by one of the matrix-free solvers provided by the Eigen library.

Afterwards, in lines 32–35, a `DiscretePotential` object is created, which evaluates the pointwise solution to the PDE from the computed Cauchy data. Therein, all required information is passed to the classes via the specialization `LaplaceSingleLayerPotential`, which is once again user defined.

5. Modules and notes on their use

Now, that a basic familiarity with a typical program flow has been established, we will introduce all Bembel modules in alphabetical order and discuss their purpose.

AnsatzSpace

The AnsatzSpace module contains the routines managing the discrete space on the surface of the geometry. Specifically, this is realized through the four classes `SuperSpace`, `Projector`, `Glue`, and `AnsatzSpace`. Therein, `SuperSpace` manages local polynomial bases on every element. Through a transformation matrix generated by the template class `Projector` which depends on the specialization of `LinearOperatorBase` and its defined traits, the `SuperSpace` can be related to B-Spline bases on every patch. To build conforming spaces (in the case of `DifferentialForm::DivergenceConforming` through continuity of the normal component across patch interfaces, in the case of `DifferentialForm::Continuous` through global C^0-continuity), the template class `Glue` assembles another transformation matrix to identify degrees of freedom across edges. Then, a coefficient vector in the `SuperSpace` can be related to one of the smooth B-Spline bases, as explained in [9,17].

ClusterTree

The ClusterTree module introduces a cluster hierarchy on parametric surfaces. The contained `ClusterTree` class takes a `Geometry` and introduces an element structure on it, in the form of an `ElementTree`. Each `ElementTreeNode` of the `ElementTree` corresponds to an entire parametric mapping (at the trees first level) or to a sub element induced by a recursive refinement strategy. The leaves of the tree correspond to the elements on which the `SuperSpace` introduces shape functions.

DuffyTrick

The DuffyTrick module provides quadrature routines for (nearly) singular integrals. Therein, the implementation is as described in the appendix of [14], essentially being an adaptation of the so-called *Sauter-Schwab quadrature rules* [21]. These routines, together with the Quadrature module, can easily be used as a standalone module on any type of quadrilateral discretization.

DummyOperator

This module provides a specialization of `LinearOperatorBase` for testing.

Geometry

The Geometry module provides the functionality required for a parametric geometry description in a computational framework. The `Geometry` class is the interface between geometry description and the remainder of the code. We provide an implementation of NURBS discretized patches via the `Patch` class, but the code is easily extensible to other parametric descriptions mapping from $[0,1]^2$ to parts of the geometry, as long as the corresponding methods for point evaluation and evaluation of the pointwise Jacobian are implemented. The Geometry module depends only on the Spline module and can be used standalone for the implementation of custom numerical codes.

H2Matrix

The H2Matrix module provides functionality for an efficient compression of the system matrix and reduction of the computational cost of the matrix–vector multiplication. Therein, the implemented algorithms correspond to the ones presented in [7, 16].

Helmholtz

The Helmholtz module provides specializations to solve Helmholtz problems.

IO

The IO module provides input–output functionality, including routines for VTK file export, timing, and writing log files.

Laplace

The Laplace module provides specializations to solve Laplace problems.

LinearForm

The `LinearForm` template class must be specialized for the assembly of the right hand side. Exemplarily, Bembel provides an implementation of the scalar `DirichletTrace` and the vector-valued `RotatedTangentialTrace` required to solve Laplace and Helmholtz Dirichlet problems and the electric field integral equation. Other specializations implementing different approaches can be easily implemented by appropriate modifications within the corresponding header files.

LinearOperator

Specializations of the `LinearOperatorBase` class govern the behaviour of most other modules. To provide a valid specialization, methods for evaluation of the integrand, i.e., including the test functions, must be provided. Moreover, the corresponding specialization of `LinearOperatorTraits` must be provided, allowing other classes to determine crucial properties such as the numerical type of the problem (in general `double` or `std::complex<double>`) and the type of discretization, i.e., either `DifferentialForm::Continuous`, corresponding to a discrete subspace of $H^{1/2}$, `DifferentialForm::DivConforming`, corresponding to a discrete subspace of $\boldsymbol{H}_{\times}^{-1/2}(\mathrm{div}_\Gamma, \Gamma)$, or `DifferentialForm::Discontinuous`, corresponding to a discrete subspace of $H^{-1/2}(\Gamma)$.

Maxwell

The Maxwell module example specializations required to solve problems based on the electric wave equation.

Potential

The Potential module introduces means to evaluate the solution to a PDE via a suitable integral operator taking the unknown of the linear system as input. It relies, once again, on a suitable specialization corresponding of the `LinearOperatorBase` class.

Quadrature

The Quadrature module provides quadrature routines for $[0,1]^n$ for $n = 1, 2$. The implementation is for arbitrary quadrature orders, where for higher-orders the required data is generated by the solution of the corresponding three term recurrence. The Quadrature module can be used standalone.

Spline

The Spline module provides basic routines related to spline function and local polynomials. The polynomials are implemented through template recursion, and should be interfaced through the `SuperSpace` class if used within Bembel. The Spline module is independent of other modules and can easily be used as the basis of other numerical codes.

6. User specific implementation

In all of the above, the user needs to provide certain implementations in order to specify custom operators. These are the routines for the evaluation of the integrand, as well as the definition of certain traits. We will briefly discuss the implementation of the Laplace single layer case, following the previous code example.

```
class LaplaceSingleLayerOperator;

template <>
struct LinearOperatorTraits<
    LaplaceSingleLayerOperator> {
  typedef Eigen::VectorXd EigenType;
  typedef double Scalar;
  enum {
    OperatorOrder = -1,
    Form = DifferentialForm::Discontinuous,
    // ...
  };
};
```

This is an example of `LinearOperatorTraits`. They pose a way for core algorithms to deduce types, such as complex or real values, and matrix formats associated with the problem to be solves. The `OperatorOrder` enum enables the core routines to choose an appropriate quadrature degree, and the `Form` designates that the basis is to be assembled as a discretization of $H^{-1/2}(\Gamma)$, i.e., discontinuous across patch boundaries. Further traits designate information for the `H2Matrix` compression, for a discussion of these, we refer to the Doxygen documentation of the code.

For an implementation of the matrix assembly, so-called `SurfacePoint`s are passed as an input. A `SurfacePoint p` can be handled as follows.

```
auto s = p.segment<2>(0);
auto w = p(2);
auto f = p.segment<3>(3);
auto f_dx = p.segment<3>(6);
auto f_dy = p.segment<3>(9);
```

The `SurfacePoint` is a `typedef` for an `Eigen::Matrix<double,Eigen::Dynamic,12>`, where the first two components `s` hold the coordinate in the reference element, the third component `w` an optional (quadrature) weight, the fourth to sixth entry `f` the corresponding point on the geometry in $\mathbb{R}^3$, and the remaining components `f_dx`

and `f_dy` the tangential vectors corresponding to differentiation in x and y direction. A surface point can be "unwrapped" via a code snipped as above, generating references to the segments of the vector.

Now, an implementation of `evaluateIntegrand_impl` could look as follows.

```
1
2  class LaplaceSingleLayerOperator
3      : public LinearOperatorBase<
       LaplaceSingleLayerOperator> {
4   public:
5    LaplaceSingleLayerOperator() {}
6    template <class T>
7    void evaluateIntegrand_impl(
8        const T &super_space, const SurfacePoint
        &p1, const SurfacePoint &p2,
9        Eigen::Matrix<
10           typename LinearOperatorTraits<
       LaplaceSingleLayerOperator>::Scalar,
11           Eigen::Dynamic, Eigen::Dynamic> *
       intval) const {
12     auto polynomial_degree = super_space.
       get_polynomial_degree();
13     auto polynomial_degree_plus_one_squared =
14         (polynomial_degree + 1) * (
       polynomial_degree + 1);
15     // Surface points p1 and p2 are unwrapped
16     // by assigning references as showcased
        above
17     auto x_kappa = x_f_dx.cross(x_f_dy).norm()
       ;
18     auto y_kappa = y_f_dx.cross(y_f_dy).norm()
       ;
19
20     (*intval) += super_space.basisInteraction(
       s, t) * evaluateKernel(x_f, y_f)* x_kappa
        * y_kappa * ws * wt;
21
22     return;
23   }
```

Lines 1–11 show that this is a specialization of the `LinearOperatorBase` class. In the implementation of the evaluation of the integrand, lines 13–19 merely extract information from the given input, the `SurfacePoint`s and the `SuperSpace`. The `SuperSpace` is aware of the refinement structure and yields methods for basis function evaluation. The actual implementation relevant for matrix assembly is in line 20, which directly corresponds to an evaluation of the matrix local element matrix for quadrature points and weights encoded in the `SurfacePoint`s.

Note that the implementation of a corresponding `LaplaceSingleLayerPotential` is completely analogous, and a specialization for the FMM is provided as an example in the repository as well, within the Laplace module.

7. Numerical results

Running a more elaborate version of the example code discussed above, including `for` loops of the polynomial order and the refinement level, as well as an error measurement via manufactured solution, yields the numerical results illustrated in Fig. 1, which behave as predicted by theory, cf. [7]. We remark that the visualization of the density in Fig. 1 was generated by the routines provided in the IO module. More involved numerical examples computed with previous versions of Bembel, including Helmholtz and Maxwell problems, are discussed in [7,17].

8. Impact

The implementation of boundary element methods in three spatial dimensions is a non-trivial task due to the necessary numerical evaluation of singular integrals and the required matrix compression to achieve computational efficiency. While this is already true for lowest-order implementations for the Laplace equation on boundary triangulations with flat elements, difficulties increase on isogeometric (or parametric) surfaces, isogeometric B-spline (or higher-order) boundary element spaces, and more involved electromagnetic problems. These implementation-related issues lead many people to refrain from the use of boundary element methods, even when a specific engineering problem is known to be solved best therewith. Our software package aims to provide a state-of-the-art boundary element toolbox for engineers who would like to apply competitive isogeometric boundary element methods in a black-box fashion. This allows users to freely employ boundary element methods as a tool in involved engineering applications. To the best of our knowledge, the functionality of higher-order B-spline boundary element spaces is the first open source implementation available for three dimensions.

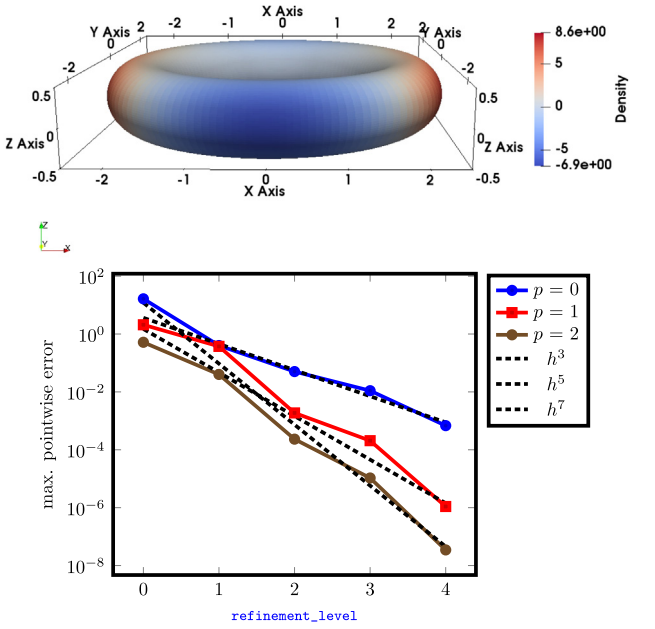

Fig. 1. Illustrations obtained from the solution of a Laplace problem. The problem specification and the code to reproduce it are in the example `example_FullLaplaceWorkflow.cpp` provided in the Bembel repository, where p corresponds to the `polynomial_degree`, and h to $(1/2)^{\texttt{refinement\_level}}$. The density is a visualization of the coefficient vector `rho`.

The major strengths of Bembel are the capability of implementing general boundary integral operators, and the direct integration into the Eigen Linear Algebra Library. This allows the user for a straightforward pre or post processing of data with a clean user interface. The dense and matrix-free algorithms of Eigen provide different kinds of solvers for the Galerkin systems or eigenvalue problems.

9. Conclusion

Bembel is an open-source library enabling users to apply isogeometric boundary element methods for potential, acoustic, electromagnetic, and many other problems in a black-box fashion. This is achieved through an easy-to-use API both for black-box use, as well as implementation of custom operators, and compatibility with the Eigen template library for linear algebra. Moreover, it incorporates state-of-the-art compression techniques for large problems, as well as OpenMP parallelization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by DFG Grants SCHO1562/3-1 and KU1553/4-1 within the project *Simulation of superconducting cavities with isogeometric boundary elements (IGA-BEM).* The work of Jürgen Dölz was partially supported by SNSF Grants 156101 and 174987, as well as the Excellence Initiative of the German Federal and State Governments and the Graduate School of Computational Engineering at TU Darmstadt. Michael Multerer was supported by SNSF Grant 137669 until 2014. The work of Felix Wolf is supported by the Excellence Initiative of the German Federal and State Governments and the Graduate School of Computational Engineering at TU Darmstadt.

Appendix A. Partial differential equations

In the following, let $\Omega \subset \mathbb{R}^3$ be a bounded domain with Lipschitz boundary $\Gamma := \partial\Omega$. Moreover, we define the exterior domain $\Omega^c := \mathbb{R}^3 \setminus \overline{\Omega}$. Bembel provides examples to the Laplace equation

$$\begin{aligned} -\Delta u_L &= 0 \quad \text{in } \Omega, \\ u_L &= g_L \quad \text{on } \Gamma, \end{aligned} \tag{LP}$$

and the Helmholtz equation

$$\begin{aligned} -\Delta u_H - \kappa_H^2 u_H &= 0 \quad \text{in } \Omega^c, \\ u_H &= g_H \quad \text{on } \Gamma, \end{aligned} \tag{HP}$$

with Sommerfeld radiation conditions towards infinity. In both cases, the boundary values have to be understood in the usual sense of traces, see, e.g. [20]. In addition, Bembel can treat the electric wave equation

$$\begin{aligned} \mathbf{curl\,curl}\, \mathbf{E}_M - \kappa_M^2 \mathbf{E}_M &= \mathbf{0} \quad \text{in } \Omega^c, \\ \mathbf{n} \times \mathbf{E}_M &= \mathbf{g}_M \quad \text{on } \Gamma, \end{aligned} \tag{MP}$$

with Silver-Müller radiation conditions towards infinity, which can be derived from Maxwell's equations. Again, the boundary values have to be understood in the sense of traces, see [19].

Appendix B. Boundary integral equations

The three boundary value problems (LP), (HP), and (MP) can each be solved by means of a *single layer potential ansatz*, i.e., setting

$$u_L(\mathbf{x}) = (\tilde{\mathcal{S}}_L \rho_L)(\mathbf{x}) := \int_\Gamma \frac{1}{4\pi \|\mathbf{x}-\mathbf{y}\|_2} \rho_L(\mathbf{y})\, d\sigma_\mathbf{y}, \qquad \mathbf{x} \in \Omega, \tag{LS}$$

$$u_H(\mathbf{x}) = (\tilde{\mathcal{S}}_H \rho_H)(\mathbf{x}) := \int_\Gamma \frac{e^{-i\kappa_H \|\mathbf{x}-\mathbf{y}\|_2}}{4\pi \|\mathbf{x}-\mathbf{y}\|_2} \rho_H(\mathbf{y})\, d\sigma_\mathbf{y}, \qquad \mathbf{x} \in \Omega^c, \tag{HS}$$

$$\begin{aligned} \mathbf{E}_M(\mathbf{x}) = (\tilde{\boldsymbol{\mathcal{S}}}_M \mathbf{j}_M)(\mathbf{x}) := &\int_\Gamma \frac{e^{-i\kappa_M \|\mathbf{x}-\mathbf{y}\|_2}}{4\pi \|\mathbf{x}-\mathbf{y}\|_2} \mathbf{j}_M(\mathbf{y})\, d\sigma_\mathbf{y} \\ &+ \frac{1}{\kappa_M^2} \nabla_\mathbf{x} \int_\Gamma \frac{e^{-i\kappa_M \|\mathbf{x}-\mathbf{y}\|_2}}{4\pi \|\mathbf{x}-\mathbf{y}\|_2} \operatorname{div}_\Gamma \mathbf{j}_M(\mathbf{y})\, d\sigma_\mathbf{y}, \\ &\mathbf{x} \in \Omega^c. \end{aligned} \tag{MS}$$

It can be shown that (LS), (HS), and (MS) solve the boundary value problems (LP), (HP), and (MP) for appropriate density functions ρ_L, ρ_H, and $\mathbf{j}_M$, see [19,20]. Taking the proper trace operators, for simplicity denoted by omiting the tilde, one arrives at the three boundary integral equations

$$\mathcal{S}_L \rho_L = g_L, \quad \mathcal{S}_H \rho_H = g_H, \quad \boldsymbol{\mathcal{S}}_M \rho_M = \mathbf{g}_M \tag{BIE}$$

on Γ to determine the unknown density functions in (LS), (HS), and (MS). We note that $\mathcal{S}_L$ and $\mathcal{S}_H$ are isomorphisms from $H^{-1/2}(\Gamma)$ to $H^{1/2}(\Gamma)$, whereas $\boldsymbol{\mathcal{S}}_M$ is an isomorphism on $\mathbf{H}_\times^{-1/2}(\operatorname{div}_\Gamma, \Gamma)$, see [19,20] for details. Thus, in view of a conforming Galerkin method, piecewise polynomial boundary element spaces are sufficient for the boundary element based solution of (LP) and (HP), whereas (MP) requires divergence conforming boundary element spaces. Unique solvability of the corresponding linear systems can be proven, see [19,20].

Remark B.1. In fact, the described single layer potential approaches solve the interior and the exterior problems of (LP), (HP), and (MP) simultaneously. Thus, $\tilde{\mathcal{S}}_H \rho_H$ and $\tilde{\boldsymbol{\mathcal{S}}}_M \mathbf{j}_M$ satisfy (HP) and (MP) also in Ω, whereas $\tilde{\mathcal{S}}_L \rho_L$ satisfies (LP) also in Ω^c, with additional radiation conditions $|u_L(\mathbf{x})| = \mathcal{O}\big(\|\mathbf{x}\|_2^{-1}\big)$ and $\|\nabla u_L(\mathbf{x})\|_2 = \mathcal{O}\big(\|\mathbf{x}\|_2^{-2}\big)$ towards infinity, see [19,20].

Appendix C. B-splines and NURBS

Let p and k be two fixed integers such that $0 \leq p < k$ and let Ξ be a locally quasi uniform knot vector with knots in $[0, 1]$, see [22]. The B-spline basis $\{b_j^p\}_{0 \leq j < k}$ is then defined by recursion as

$$b_j^p(x) = \begin{cases} \chi_{[\xi_j, \xi_{j+1})}, & \text{if } p = 0, \\ \frac{x-\xi_j}{\xi_{j+p}-\xi_j} b_j^{p-1}(x) + \frac{\xi_{j+p+1}-x}{\xi_{j+p+1}-\xi_{j+1}} b_{j+1}^{p-1}(x), & \text{else,} \end{cases}$$

where χ_M denotes the indicator function for the set M. Having the B-spline basis at our disposal, we define the spline space $S^p(\Xi) := \operatorname{span}(\{b_j^p\}_{0 \leq j < k})$.

A NURBS mapping $\boldsymbol{\gamma}_i : \square \to \mathbb{R}^3$ on the unit square $\square = [0, 1]^2$ is given by

$$\boldsymbol{\gamma}_j(x, y) := \sum_{0 \leq j_1 < k_1} \sum_{0 \leq j_2 < k_2} \frac{\mathbf{c}_{j_1,j_2} b_{j_1}^{p_1}(x) b_{j_2}^{p_2}(y) w_{j_1,j_2}}{\sum_{i_1=0}^{k_1-1} \sum_{i_2=0}^{k_2-1} b_{i_1}^{p_1}(x) b_{i_2}^{p_2}(y) w_{i_1,i_2}},$$

and described by its control points $\mathbf{c}_{j_1,j_2} \in \mathbb{R}^3$ and weights $w_{i_1,i_2} > 0$. For further concepts and algorithmic realization of the NURBS, we refer to [22].

Appendix D. Boundary representation

Bembel assumes that the boundary representations are the union of several *patches* Γ_i, i.e.,

$$\boldsymbol{\gamma}_i : \square \to \Gamma_i \quad \text{with} \quad \Gamma_i = \boldsymbol{\gamma}_i(\square) \quad \text{for } i = 1, 2, \ldots, M,$$

where $\boldsymbol{\gamma}_i$ is given by a NURBS mapping with outward pointing normal. The boundary itself is then the collection of all patches

$$\Gamma = \bigcup_{i=1}^{M} \Gamma_i,$$

where the intersection $\Gamma_i \cap \Gamma_{i'}$ consists at most of a common vertex or a common edge for $i \neq i'$.

In order to ensure conforming meshes, Bembel also imposes the following matching condition on the parametrizations: For each $\mathbf{x} = \boldsymbol{\gamma}_i(\mathbf{s})$ on a common edge of Γ_i and $\Gamma_{i'}$, there has to exist a bijective and affine mapping $\Xi : \square \to \square$ such that there holds $\boldsymbol{\gamma}_i(\mathbf{s}) = (\boldsymbol{\gamma}_{i'} \circ \Xi)(\mathbf{s})$. This means that the parameterizations $\boldsymbol{\gamma}_i$ and $\boldsymbol{\gamma}_{i'}$ coincide on the common edge except for orientation.

Appendix E. B-spline boundary element spaces

The boundary element spaces for the Galerkin method are constructed as isogeometric multi-patch B-spline spaces, see [23]. Bembel uses equidistant knot vectors $\Xi_1 = \Xi_2$ with 2^L elements and the same polynomial degrees in each direction, i.e., $p_1 = p_2$, on the unit square. The corresponding spline spaces are thus given by

$$\mathbb{S}^0_{\mathbf{p},\boldsymbol{\Xi}}(\square) = S_{p_1,\Xi_1}([0,1]) \otimes S_{p_2,\Xi_2}([0,1]),$$
$$\mathbb{S}^1_{\mathbf{p},\boldsymbol{\Xi}}(\square) = S_{p_1,\Xi_1}([0,1]) \otimes S_{p_2-1,\Xi_2'}([0,1]) \times S_{p_1-1,\Xi_1'}([0,1]) \otimes S_{p_2,\Xi_2}([0,1]),$$
$$\mathbb{S}^2_{\mathbf{p},\boldsymbol{\Xi}}(\square) = S_{p_1-1,\Xi_1'}([0,1]) \otimes S_{p_2-1,\Xi_2'}([0,1]),$$

for $\mathbf{p} = (p_1, p_2)$ and $\boldsymbol{\Xi} = (\Xi_1, \Xi_2)$. Herein, Ξ_i', for $i = 1, 2$, denotes the truncated knot vector, i.e., the knot vector without its first and last element.

Due to the representation of the computational geometry by patches, the spaces constructed on the unit square can easily be lifted to the patches on the boundary by the *Piola transform*. Enforcing continuity conditions across patch boundaries yields conforming discretizations

$$\mathbb{S}^0_{\mathbf{p},\boldsymbol{\Xi}}(\Gamma) \subset H^{1/2}(\Gamma),$$
$$\mathbb{S}^1_{\mathbf{p},\boldsymbol{\Xi}}(\Gamma) \subset \mathbf{H}_\times^{-1/2}(\mathrm{div}_\Gamma, \Gamma),$$
$$\mathbb{S}^2_{\mathbf{p},\boldsymbol{\Xi}}(\Gamma) \subset H^{-1/2}(\Gamma).$$

References

[1] Śmigaj W, Betcke T, Arridge S, Phillips J, Schweiger M. Solving boundary integral problems with BEM++. ACM Trans. Math. Software 2015;41(2):6:1–40.

[2] Hiptmair H, Kielhorn L. BETL — A generic boundary element template library. Tech. rep., ETH Zurich; 2012.

[3] Hughes TJR, Cottrell JA, Bazilevs Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Engrg. 2005;194(39):4135–95.

[4] Spink M, Claxton D, de Falco C, Vázquez R. The NURBS toolbox, http://octave.sourceforge.net/nurbs/index.html.

[5] Bingol OR, Krishnamurthy A. NURBS-python: An open-source object-oriented NURBS modeling framework in python. SoftwareX 2019;9:85–94.

[6] Aimi A, Calabrò F, Diligenti M, Sampoli M, Sangalli G, Sestini A. Efficient assembly based on B-spline tailored quadrature rules for the IgA-SGBEM. Comput. Methods Appl. Mech. Engrg. 2018;331(Supplement C):327–42.

[7] Dölz J, Harbrecht H, Kurz S, Schöps S, Wolf F. A fast isogeometric BEM for the three dimensional Laplace- and Helmholtz problems. Comput. Methods Appl. Mech. Engrg. 2018;330(Supplement C):83–101.

[8] Feischl M, Gantner G, Haberl A, Praetorius D. Optimal convergence for adaptive IGA boundary element methods for weakly-singular integral equations. Numer. Math. 2017;136:147–82.

[9] Marussig B, Zechner J, Beer G, Fries T. Fast isogeometric boundary element method based on independent field approximation. Comput. Methods Appl. Mech. Engrg. 2015;284(0):458–88.

[10] Simpson RN, Bordas S, Trevelyan J, Rabczuk T. A two-dimensional isogeometric boundary element method for elastostatic analysis, Vol. 209–212. 2012, p. 87–100.

[11] Taus M. Isogeometric analysis for boundary integral equations (Ph.D. thesis), University of Texas at Austin; 2015.

[12] Takahashi T, Matsumoto T. An application of fast multipole method to isogeometric boundary element method for Laplace equation in two dimensions. Eng. Anal. Bound. Elem. 2012;36(12):1766–75.

[13] Dölz J, Harbrecht H, Kurz S, Multerer M, Schöps S, Wolf F. Bembel. 2019, http://dx.doi.org/10.5281/zenodo.2671596, http://www.bembel.eu.

[14] Harbrecht H. Wavelet Galerkin schemes for the boundary element method in three dimensions (Ph.D. thesis), Technische Universität Chemnitz; 2001.

[15] Harbrecht H, Peters M. Comparison of fast boundary element methods on parametric surfaces. Comput. Methods Appl. Mech. Engrg. 2013;261–262:39–55.

[16] Dölz J, Harbrecht H, Peters M. An interpolation-based fast multipole method for higher-order boundary elements on parametric surfaces. Internat. J. Numer. Methods Engrg. 2016;108(13):1705–28.

[17] Dölz J, Kurz S, Schöps S, Wolf F. Isogeometric boundary elements in electromagnetism: rigorous analysis, fast methods, and examples. SIAM J. Sci. Comput 2019;41(5):B983–1010.

[18] Guennebaud G, Jacob B, et al. Eigen v3. 2010, http://eigen.tuxfamily.org.

[19] Buffa A, Hiptmair R. Galerkin boundary element methods for electromagnetic scattering. In: Ainsworth M, Davies P, Duncan D, Rynne B, Martin P, editors. Topics in computational wave propagation. Springer; 2003, p. 83–124.

[20] Steinbach O. Numerical approximation methods for elliptic boundary value problems. New York: Springer Science & Business; 2008.

[21] Sauter SA, Schwab C. Quadrature for *hp*-Galerkin BEM in $\mathbb{R}^3$. Numer. Math. 1997;78(2):211–58.

[22] Piegl L, Tiller W. The NURBS book. second ed.. Springer; 1997.

[23] Buffa A, Dölz J, Kurz S, Schöps S, Vázquez R, Wolf F. Multipatch approximation of the de Rham sequence and its traces in isogeometric analysis. Numer. Math. 2020;144:201–36.