The Wavelet Galerkin Method for the Polarizable Continuum Model in Quantum Chemistry

Inauguraldissertation

zur

Erlangung der Würde eines Doktors der Philosophie

vorgelegt der

Philosophisch-Naturwissenschaftlichen Fakultät

der Universität Basel

von

Mihaela Monica Bugeanu

aus

Bukarest, Rumänien

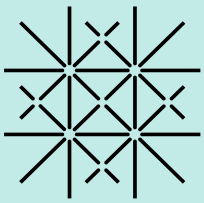
Basel, 2017

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Helmut Harbrecht
Prof. Dr. Benjamin Stamm

Basel, 19. September, 2017

Prof. Dr. Martin Spiess
Dekan



Universität Basel

The Wavelet Galerkin Method for the Polarizable Continuum Model in Quantum Chemistry

PhD Thesis

Monica Bugeanu

December 17, 2017

Supervisor
Prof. Dr. Helmut Harbrecht

Co-Examiner
Prof. Dr. Benjamin Stamm

Acknowledgements

The glorious day has finally arrived: this is the cherry on top of my dissertation. This period has had a great impact on me, not only on an academic level, but also on a personal one and I would like to give thanks to the people who supported and helped me to get to where I am today.

My gratitude goes to my PhD supervisor Prof. Dr. Helmut Harbrecht, for the tremendous amount of patience that he has shown during the past five years. He has guided me through the deep waters of the PhD period with a soft hand and showed immense knowledge, motivation and understanding.

I would also like to thank my co-examiner, Prof. Dr. Benjamin Stamm, for reviewing and evaluating my work, and moving mountains to accommodate for the best possible defense date.

I thank my fellow group members, current and former, for the stimulating discussions and for the fun we had also outside the walls of the university. For the time at the beginning I would like to thank Michael Peters, Markus Siebenmorgen and Manuela Utzinger for helping me through the "infant" period of my work and Jürgen Dölz for the patience when describing spheres and squares. I feel like I have truly gained a second family. A very special thanks goes also the the administration of the institute, this place would fall apart without You.

This work hast emerged from a collaboration with the department of theoretical chemistry in Tromsø and I would like to thank Prof. Dr. Luca Frediani and Roberto DiRemigio for the great time spent in the north and the fruitful discussions about chemistry.

Special thanks goes also to Manuela Utzinger and Nikolai Ostapowicz who have taken the time to proof read my thesis.

Last but not least, I would like to thank my family and my boyfriend for their wise counsel, always having a sympathetic ear, and being there for me.

Thank you very much, everyone!

Monica Bugeanu

Basel, December 17, 2017.

Contents

Abstract

The polarizable continuum model (PCM) is a well established method for computing solvation effects. Its attractiveness comes from the fact that the solution is modelled as a continuum instead of modelling each atom individually. This allows for more complex simulations. Nevertheless, there are still a couple of challenges to face. One of them being the computational time required at the limit of large systems, or when high accuracy is needed.

The wavelet boundary element method can be used to overcome these problems, provided a reliable cavity generator is used. The PCM calculates the molecular free energy in solution as the sum of electrostatic, dispersion-repulsion and the cavitation energy by solving the underlying partial differential equations. Those differential equations can be transformed into integral equations, solely defined on the boundary of the molecular cavity, by applying the integral equation formalism (IEFPCM). The integral equations can then be discretized using the wavelet boundary element method. The resulting sparse system of linear equations can be solved reliably by iterative solvers, leading to high accuracy solutions even for large systems.

The challenge of this approach lies in the generation and refinement of the molecular cavities on which the interactions take place. Since wavelets are defined as a tensor product of one-dimensional functions, the wavelet boundary element method needs a discretization into quadrangular patches of the molecular cavity. This feature is missing in common commercial mesh generation tools, which focus on generating triangular meshes. Making use of characteristic functions for defining which points are inside the cavity, a model for generating the molecular surface independently of its exact description is achieved. We present here a way for generating triangular meshes which are subsequently merged into quadrangular ones. We apply geometrical quantities that measure the fitness of the shapes involved and improve the position of individual points iteratively. Thereby, we generate quadrangular meshes with an unprecedented quality, which ultimately can be refined in a hierarchical manner for the use of wavelets.

Overall, this method can generate reliable parametrizations on a variety of smooth cavities. With the help of the wavelet boundary element method, the PCM equations can thus be solved with high accuracy also on large systems.

CHAPTER 1

Solving the Schrödinger equation

1.1. Schrödinger equation

The main focus of computational chemistry is to characterize atoms and compounds by computing energy and structures that result from the interaction between electrons and nuclei. The underlying equation that describes this interaction for a time-independent setting is the Schrödinger equation,

$$H\Psi(\boldsymbol{r}, \boldsymbol{R}) = E\Psi(\boldsymbol{r}, \boldsymbol{R}), \tag{1.1}$$

with Ψ being the wave function which fully describes the system, E the ground state energy, $\boldsymbol{r}$ the position of the electrons, and $\boldsymbol{R}$ the position of the nuclei. This equation is known since 1926, [65]. The Hamiltonian, H, for a system of N_e electrons and N_n nuclei in a non-relativistic setting looks as follows:

$$\begin{aligned} H = &- \sum_{j=1}^{N_e} \frac{\langle \boldsymbol{p}_j, \boldsymbol{p}_j \rangle}{2m} - \sum_{\alpha=1}^{N_n} \frac{\langle \boldsymbol{P}_\alpha, \boldsymbol{P}_\alpha \rangle}{2M_\alpha} \\ &- \sum_{j=1}^{N_e} \sum_{\alpha=1}^{N_n} \frac{Z_\alpha e^2}{\|\boldsymbol{r}_j - \boldsymbol{R}_\alpha\|} + \sum_{j=1}^{N_e} \sum_{k=1}^{j-1} \frac{e^2}{\|\boldsymbol{r}_j - \boldsymbol{r}_k\|} + \sum_{\alpha=1}^{N_n} \sum_{\beta=1}^{\alpha-1} \frac{Z_\alpha Z_\beta e^2}{\|\boldsymbol{R}_\alpha - \boldsymbol{R}_\beta\|}. \end{aligned} \tag{1.2}$$

Here, the electrons are described by indices j and k, mass m, position $\boldsymbol{r}_{j/k}$, momentum operator $\boldsymbol{p}_{j/k} = -i\hbar\nabla_{j/k}$, and the electronic charge e. The nuclei are described by indices α and β, atomic number $Z_{\alpha/\beta}$, mass $M_{\alpha/\beta}$, position $\boldsymbol{R}_{\alpha/\beta}$, and momentum operator $\boldsymbol{P}_{\alpha/\beta} = -i\hbar\nabla_{\alpha/\beta}$. Inserting the momentum operators into the Hamiltonian as given in eq. (1.2) leads to a second order differential equation. The first two terms,

$$\begin{aligned} T_e(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_e}) &= -\sum_{j=1}^{N_e} \frac{\langle \boldsymbol{p}_j, \boldsymbol{p}_j \rangle}{2m}, \\ T_n(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{N_n}) &= -\sum_{\alpha=1}^{N_n} \frac{\langle \boldsymbol{P}_\alpha, \boldsymbol{P}_\alpha \rangle}{2M_\alpha}, \end{aligned} \tag{1.3}$$

contain the kinetic energy of the electrons and the nuclei, respectively. The third term,

$$V_{ne}(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_e}, \boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{N_n}) = -\sum_{j=1}^{N_e} \sum_{\alpha=1}^{N_n} \frac{Z_\alpha e^2}{\|\boldsymbol{r}_j - \boldsymbol{R}_\alpha\|}, \tag{1.4}$$

represents the electron-nucleus attraction. The last two terms are the electron-electron

$$V_{ee}(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_e}) = \sum_{j=1}^{N_e} \sum_{k=1}^{j-1} \frac{e^2}{\|\boldsymbol{r}_j - \boldsymbol{r}_k\|}, \tag{1.5}$$

and the nucleus-nucleus repulsions,

$$V_{nn}(\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{N_n}) = \sum_{\alpha=1}^{N_n} \sum_{\beta=1}^{\alpha-1} \frac{Z_\alpha Z_\beta e^2}{\|\boldsymbol{R}_\alpha - \boldsymbol{R}_\beta\|}. \tag{1.6}$$

Eq. (1.1) with the Hamiltonian as given in eq. (1.2) theoretically solves every system one might be interested in. However, the physicist Paul Dirac noticed already in 1929 the need for approximate methods, [17], since solving this equation ab-initio, i.e. modelling each electron and nucleus individually, leads to very large systems that would have to be tackled.

For example, even solving the full Schrödinger equation for a single iron atom, including all its 26 electrons and 26 protons in three dimensions, would lead to a wave function of 156 variables. If one would discretize the space by a crude hypercube with only 10 points per unknown, one would need to work with 10^{156} spatial variables in the resulting system. This illustrates that even solving the iron atom by using the full Hamiltonian from eq. (1.2) can be problematic and suitable approximations are needed.

In order to reduce the complexity of the Hamiltonian, the separation of variables could be used. A split of the Hamiltonian in form of $H(\boldsymbol{r}, \boldsymbol{R}) = H_1(\boldsymbol{r}) + H_2(\boldsymbol{R})$ is thus sought after. In case such a split can be found, it would lead to a separation in the wave function and the energy as follows

$$\begin{aligned} \Psi(\boldsymbol{r}, \boldsymbol{R}) &= \psi_1(\boldsymbol{r})\psi_2(\boldsymbol{R}), \\ E &= E_1(\boldsymbol{r}) + E_2(\boldsymbol{R}). \end{aligned}$$

The energies $E_1(\boldsymbol{r})$ and $E_2(\boldsymbol{R})$ would be given by the eigenvalues of the decoupled system

$$H_1(\boldsymbol{r})\ \psi_1(\boldsymbol{r})\ = E_1(\boldsymbol{r})\ \psi_1(\boldsymbol{r}),$$
$$H_2(\boldsymbol{R})\psi_2(\boldsymbol{R}) = E_2(\boldsymbol{R})\psi_2(\boldsymbol{R}).$$

Unfortunately, the potential energy, V_{ne}, found in eq. (1.4) couples the nuclei and the electrons in the Hamiltonian and prohibits its exact separation of variables.

1.2. Born-Oppenheimer approximation

The first approximation that can be used to simplify the Hamiltonian of the Schrödinger eq. (1.2) is the so-called Born-Oppenheimer approximation, where one fixes the position of the nuclei. The argument behind this approximation is the fact that the electron mass is about three orders of magnitude smaller than that of a proton or a neutron and thus the movement of the nuclei happens much slower than that of the electrons. Due to their difference in size and speed, one can even assume that the electrons are always in the ground state configuration for all possible positions of the nuclei. The wave function can hence be separated into a purely electronic part and a part only containing the nuclear coordinates:

$$\Psi(\boldsymbol{r}, \boldsymbol{R}) = \psi_n(\boldsymbol{R})\ \psi_e(\boldsymbol{r}).$$

This introduces also a split in the Hamiltonian into an electronic Hamiltonian, H_e, and a nuclear part, H_n, given by

$$H_e = -\sum_{j=1}^{N_e} \frac{\langle \boldsymbol{p}_j, \boldsymbol{p}_j \rangle}{2m} - \sum_{j=1}^{N_e}\sum_{\alpha=1}^{N_n} \frac{Z_\alpha e^2}{\|\boldsymbol{r}_j - \boldsymbol{R}_\alpha\|} + \sum_{j=1}^{N_e}\sum_{k=1}^{j-1} \frac{e^2}{\|\boldsymbol{r}_j - \boldsymbol{r}_k\|},$$
$$H_n = -\sum_{\alpha=1}^{N_n} \frac{\langle \boldsymbol{P}_\alpha, \boldsymbol{P}_\alpha \rangle}{2M_\alpha} + \sum_{\alpha=1}^{N_n}\sum_{\beta=1}^{\alpha-1} \frac{Z_\alpha Z_\beta e^2}{\|\boldsymbol{R}_\alpha - \boldsymbol{R}_\beta\|}.$$

The only term in the electronic Hamiltonian, H_e, that depends on the position of the nuclei is the potential term $V_{ne}(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_{N_e}, \boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_{N_n})$ from eq. (1.4). This dependency is solved by keeping the nuclei fixed while computing the electronic part of the wave function and solving the nuclear part in a later step.

The energy calculated using the electronic Hamiltonian is called the *purely electronic energy*. In order to get the energy of the electrons for a given configuration of the nuclei, one has to add the potential energy generated by the nuclei, given by eq. (1.6). The kinetic energy of the nuclei, T_n, defined in eq. (1.3) can be ignored in this step, since the nuclei are considered fixed in the Born-Oppenheimer approximation.

Solving the electronic Hamiltonian for several positions of the nuclei results in a potential energy surface (PES), where critical points can be used to find equilibrium and transition states. To arrive at the molecular energy for a given configuration, the correction term stemming from the kinetic energy of the nuclei has to be computed.

In the following, further simplifications and approximations will be introduced to reduce the complexity of the calculations even more.

1.3. Ab-initio methods

This section is concerned with approximation methods which calculate the wave function not by relying on experimental data, but rather on mathematical approximations, also called ab-initio methods, like the Born-Oppenheimer approximation mentioned earlier.

1.3.1. Hartree-Fock approximation

After fixing the position of the nuclei according to the Born-Oppenheimer approximation, there is still a lot of interaction in the electronic Hamiltonian, H_e, stemming from the electron-electron repulsion from eq. (1.5).

One can replace this such that each electron feels the presence of all other electrons in an average way. This can be combined with the influence of the nuclei in an electric field acting in radial direction equally on all electrons. The electronic wave function can now be written as a combination of Slater determinants of one-electron wave functions. The Slater determinant guarantees the antisymmetry condition, meaning that switching the positions of two electrons within one Slater determinant leads to a change in sign for the wave function. The antisymmetry condition is a postulate of quantum mechanics, which has the Pauli exclusion principle as a consequence. The Pauli exclusion principle states that no two electrons can occupy the same quantum state as postulated by Pauli in [53].

The Hartree-Fock approximation, also called the self-consistent field approximation, requires that the estimation of the central field is *self-consistent*, as described by Fock, Hartree, and Slater in [18, 32, 68]. The central field is the combined electric field of the electrons and nuclei acting in a radial direction. This means that the central field, calculated by using a charge distribution, has to be equivalent to the assumed central field used for calculating that charge distribution. In practice, this is done iteratively. The central field is used as an input for solving the electronic Hamiltonian and calculating a new charge distribution. The charge distribution in turn yields a new value for the central field. This is done until a stationary state is reached.

Several improvements can be made in order to ensure the convergence of the fix-point iteration. One of them is to use a relaxation of the one-electron wave functions. This means that the wave function is a linear combination of the wave function from the previous iteration and the newly calculated wave function. Another method is to first calculate the wave function for a positive ion, having an increased nuclear charge, and to use the results of this simulation as a starting point for the neutral molecule. For all the details, see [12, 36, 76] and the references therein.

1.3.2. Post Hartree-Fock methods

The Hartree-Fock method constructs the multi-electron wave function by a linear combination of non-interacting electrons. The main idea of post Hartree-Fock methods is

to include corrections for the correlation of the electrons.

The configuration interaction (CI) method accounts for the electron interaction by expanding the Hartree-Fock Hamiltonian in a different basis of wave functions, called the configuration state functionals. These are linear combinations of spin orbitals and describe the electron movement from the occupied states to the excited states. Applying these functionals leads to the CI method. If only functionals that move one electron are considered, the configuration interaction singles (CIS) method arises, which can give a better approximation to the excited states of the molecule without changing the ground state energy. Taking also double excitations into account, the configuration interaction singles doubles (CISD) would arise. It would lead to a different ground state energy which takes into account the corrections stemming from doubly excited states. If all interactions are considered, the method recovers the solution of the exact Hamiltonian from eq. (1.2) and is called full-CI. For more details, see [76] and the references therein.

Starting with the Hartree-Fock solution, one could also construct multi-electron wave functions that model correlations by inserting a cluster operator. This is the idea behind the coupled cluster (CC) method as described in [8] for molecules and atoms. The wave function is herein represented as the coupled cluster operator, acting on the ground state by an exponential ansatz

$$\Psi = e^T \Phi_0,$$

where T is the coupling operator and Φ_0 is the Hartree-Fock solution. The coupled cluster method guarantees size extensivity of the system by construction, which is not the case for the truncated CI methods. Size extensivity means that the energy of a non-interacting system, for example of helium atoms, He, scales directly with the number of the particles as follows:

$$E(N\,\mathrm{He}) = NE(\mathrm{He}).$$

In a similar fashion as for the CI method, the CC method can account for single (CCS), double (CCSD), triple (CCSDT), and quadruple (CCSDTQ) excitations depending on the operator used. The CCSDT method performs at a similar level as the full-CI method. This also includes size consistency, which means that the energy of a combined system AB and the individual systems far apart should satisfy

$$E(A) + E(B) = E(AB).$$

An example would be the H_2 molecule and the energy of two hydrogen atoms. However, this equation does not hold for the standard Hartree-Fock method, see [56] for details.

Both methods, CI and CC, recover the solution of the exact Hamiltonian if all excitations are considered. The complexity, however, grows for both methods like $O(N!)$ with the number of particles, N. For a limited number of excitations implemented, the CC has approximately the same complexity, but higher accuracy.

1.4. Density functional theory

A different class of methods avoids calculating the wave function altogether. Hohenberg and Kohn, [34], received the Nobel prize for their proof that shows that the electronic

ground state energy entirely depends on the electron density,

$$\rho_e = N_e \int \psi_e^\star(\boldsymbol{r}, \boldsymbol{r}_2, \ldots \boldsymbol{r}_{N_e}) \psi_e(\boldsymbol{r}, \boldsymbol{r}_2, \ldots \boldsymbol{r}_{N_e}) \, \mathrm{d}\boldsymbol{r}_2 \, \mathrm{d}\boldsymbol{r}_3 \ldots \mathrm{d}\boldsymbol{r}_{N_e}.$$

This results in an energy functional depending only on the electron density,

$$E_0 = E[\rho_e].$$

The following simple geometric explanation is found in [36]. The integral of the density defines the number of electrons, the cusps in the density are located at the position of the nuclei, and the heights of the cusps are proportional to the nuclear charge. These cusps stem from the Coulomb potential. One only has to calculate the density distribution, $\rho(\boldsymbol{x})$, as a function of space instead of calculating the wave function which depends on the $\boldsymbol{r}, \boldsymbol{p}$ and $\boldsymbol{R}, \boldsymbol{P}$ vector variables, the position and the momentum of the particles. Since the exact functional of the density is unknown, the Kohn-Sham approximation is used, where the density of the complete system is replaced by a system of non-interacting particles that generate the same density as any given system of interacting particles. The single particle density is expressed in terms of a wave function expanded in a basis set of orbitals. Several basis functions can be used and they are chosen according to the system at hand and the properties of interest.

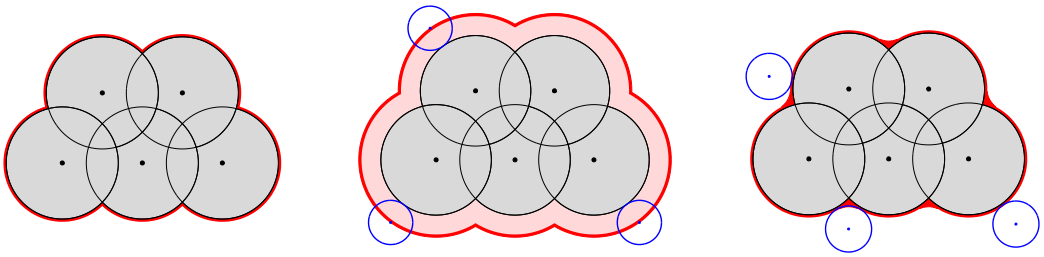
The simplest density functional methods rely only on the electron density like the local density approximation (LDA), [40], or the local spin density approximation (LSDA), [37]. The inclusion of the gradient of the electron density, leads to the group of the generalized gradient approximation (GGA) methods, [3]. There is also the group of meta-GGA methods which include higher order derivatives, [72]. The so-called hybrid methods consider the exchange correlation in a Hartree-Fock manner, for example by using the B3LYP functional as in [4].

1.5. Continuum models

Many reactions in chemistry take place in solution, leading to systems with a high number of interacting particles. The previously described methods fail here, due to the large number of particles involved. The idea behind the continuum models is to include the solvent as a continuum which is characterized by its dielectric constant. The interaction is then restricted to the surface of the molecule, also called the molecular cavity. The description of the cavities used in practice is found in sec. 1.5.1. In the following, the model will be explained for a molecule, a solute, surrounded by a solvent.

1.5.1. Types of cavities used in continuum models

There are three types of cavities that play a role in molecular chemistry: the van der Waals surface, the solvent accessible surface, and the Connolly surface, also known as the solvent excluded surface. All these cavities are schematically shown in fig. 1.7.



(a) Van der Waals surface. **(b)** Solvent accessible surface. **(c)** Connolly surface.

Figure 1.7. Types of cavity surfaces that can be encountered.

The cavity construction starts with the atoms of the molecule, defined by their centres and their van der Waals radii. This already defines the van der Waals surface (VWS), seen in fig. 1.7a, which is the visible surface generated by the interlocking spheres of the molecule. It is the most common surface encountered in chemistry packages and is exactly generated by the discretization of the visible part of the atoms, resulting in convex spherical partial surfaces. Nonetheless, this surface does not take into account the solvent.

The simplest cavity that does take the solvent into account is the solvent accessible surface (SAS), [45], depicted in fig. 1.7b. It is easily obtained by adding the radius of the solvent to the radii of the atoms. The surface that is generated this way can also be described analytically by the visible convex spherical partial surfaces, as for the VWS.

Although the SAS takes into account some radius characterizing the solvent, a better description of the surface at which the interactions take place is the Connolly surface, also called solvent excluded surface (SES), [10]. The SES can be seen in fig. 1.7c. This surface is generated by rolling a probe sphere, with the radius depending on the solvent, over the VWS. The visible partial surfaces are described not only by parts of the atoms, but also by two additional partial surface types coming from the smoothing property of rolling the probe sphere over the molecule. These partial surface types will be presented later in chpt. 4 alongside with an algorithm that focuses on generating a parametrization of the SES in a reliable and molecule-independent fashion.

An overview of existing algorithms for generating the cavity is presented in sec. 2.5.

1.5.2. Poisson equation

The most crude approximation used in continuum models considers only the classical electrostatic potential as a function of the charge density of the molecule. This results in solving the Poisson equation for a medium with dielectric constant ε:

$$-\Delta u(\boldsymbol{x}) = \frac{4\pi\rho(\boldsymbol{x})}{\varepsilon}. \tag{1.8}$$

The solute is represented explicitly by the charge density distribution, $\rho(\boldsymbol{x})$, which is constrained to the molecular cavity, while the solvent is described implicitly by the dielectric constant, ε. Methods like the Poisson-Boltzmann method solve the equation

both inside and outside the cavity, as described in [36], by setting the dielectric constant ε equal to $2-4$ inside the cavity and the charge distribution $\rho(\boldsymbol{x}) = 0$ outside of the cavity.

1.5.3. Self-consistent reaction field methods

An improvement would be to use self-consistent reaction field models where the problem is split into a chemically relevant part and a continuum part. A loop is then used until self-consistency is reached. Close to the solute, the problem is solved by using a quantum mechanical model, for example, any of the aforementioned approximations. Therein, the electrostatic potential, $u(\boldsymbol{x})$, from the Poisson eq. (1.8) is taken into account. The resulting charge distribution, $\rho(\boldsymbol{x})$, enters in a new continuum solution which results in a new electrostatic potential, $u(\boldsymbol{x})$. This is then repeated until self-consistency is reached.

Self-consistent reaction field models can be further differentiated, according to the splitting of the quantum and the continuum problem, between methods that still take into consideration the atomistic effects of the solvent next to the solute and methods where the solution is an entirely homogeneous medium. Among the first type of methods, one encounters molecular mechanics, [13, 66], where the molecules of the solvent follow the classical mechanical laws, or polarizable embedding models, [48], where a hybrid model of quantum mechanics and molecular mechanics is used. In the polarizable embedding models, a small number of molecules close to the solvent, called the first solvation shell, are treated quantum mechanically and molecules that have a smaller influence are treated only mechanically or as an average influence. The polarizable continuum model (PCM), which was introduced by Miertuš and Tomasi in 1981, [47], disregards the atomistic structure of the solvent altogether and replaces it by a fluid characterized by the dielectric constant. This model will be described in detail in the next section.

1.6. Polarizable continuum model

As mentioned in the previous section, the PCM replaces the solution around the solvent by a continuous infinite medium, which is characterized only by a couple of parameters. The interaction of the solvent with the solute takes place only on the boundary of the molecular cavity. It then solves for the electrostatic potential, $u(\boldsymbol{x})$, taking into account the solute charge density, $\rho(\boldsymbol{x})$, and the infinite solvent. Three different cases can be modelled using the PCM according to the parameters used in describing the equations for the electrostatic potential.

1.6.1. Boundary integral operators

1.6.1.1. Neutral solutions

The most common case where PCM is used is the case of a neutral solution characterized by a constant permittivity. The governing equations for the electrostatic potential of a charge distribution, $\rho(\boldsymbol{x})$, which is supported inside the cavity, Ω, are

$$\begin{aligned} -\Delta u_i(\boldsymbol{x}) &= \rho(\boldsymbol{x}) && \text{in } \Omega, \\ -\varepsilon\Delta u_e(\boldsymbol{x}) &= 0 && \text{in } \Omega^c := \mathbb{R}^3 \setminus \overline{\Omega}, \\ u_i(\boldsymbol{x}) = u_e(\boldsymbol{x}),\ \langle \nabla u_i(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle &= \langle \varepsilon\nabla u_e(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle && \text{on } \Gamma := \partial\Omega, \\ |u_e(\boldsymbol{x})| &= \mathcal{O}\big(\|\boldsymbol{x}\|^{-1}\big) && \text{as } \|\boldsymbol{x}\| \to \infty, \end{aligned} \tag{1.9}$$

with Γ being the boundary of the molecular cavity, $\rho(\boldsymbol{x})$ the charge density distribution inside the cavity, and ε the dielectric constant of the solvent. The two equations on the molecule's boundary Γ represent the continuity requirement of the electrostatic potential across the boundary taken from the inside, $u_i(\boldsymbol{x})$, and from the outside, $u_e(\boldsymbol{x})$, and the jump condition in normal direction of the potential, [35]. The last equation states that the electrostatic potential is zero at infinity to guarantee uniqueness of the solution.

In order to solve the eq. (1.9) for the electrostatic potential, one can make use of many methods for partial differential equations. One of the most prominent methods is to transform the partial differential equation into an integral equation defined on the boundary of the cavity. In order to achieve that, one can apply Green's formula and ends up with the integral equation formulation for the PCM (IEFPCM). This results in utilizing the single and double layer operators for the interior and exterior Poisson equation to transport the problem to the boundary, as described in [7].

For any problem where the fundamental solution is known, one can write the potential as an evaluation of the single and double layer operators as follows:

$$\begin{aligned} u(\boldsymbol{x}) &= (\mathcal{V}_i\langle\nabla u_i, \boldsymbol{n}\rangle)(\boldsymbol{x}) - (\mathcal{K}_i u_i)(\boldsymbol{x}) + \mathcal{N}_\rho(\boldsymbol{x}) && \text{in } \Omega, \\ u(\boldsymbol{x}) &= (\mathcal{V}_e\langle\varepsilon\nabla u_e, \boldsymbol{n}\rangle)(\boldsymbol{x}) - (\mathcal{K}_e u_e)(\boldsymbol{x}) && \text{in } \Omega^c. \end{aligned} \tag{1.10}$$

Moving to the boundary, $\boldsymbol{x} \to \Gamma$, leads to the interior and the exterior Dirichlet-to-Neumann maps:

$$\begin{aligned} (\mathcal{V}_i\langle\nabla u_i, \boldsymbol{n}\rangle)(\boldsymbol{x}) &= \left(\left(\mathcal{K}_i + \frac{1}{2}\right) u_i\right)(\boldsymbol{x}) - \mathcal{N}_\rho(\boldsymbol{x}), \\ (\mathcal{V}_e\langle\varepsilon\nabla u_e, \boldsymbol{n}\rangle)(\boldsymbol{x}) &= \left(\left(\mathcal{K}_e - \frac{1}{2}\right) u_e\right)(\boldsymbol{x}). \end{aligned} \tag{1.11}$$

Here, the interior single layer operator $\mathcal{V}_i$ and the interior double layer operator $\mathcal{K}_i$, taken on the boundary, are given by:

$$\left.\begin{aligned} (\mathcal{V}_i u)(\boldsymbol{x}) &= \int_\Gamma \frac{u(\boldsymbol{y})}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|}\, \mathrm{d}\sigma_{\boldsymbol{y}} \\ (\mathcal{K}_i u)(\boldsymbol{x}) &= \int_\Gamma \frac{\langle \boldsymbol{n}(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|^3} u(\boldsymbol{y})\, \mathrm{d}\sigma_{\boldsymbol{y}} \end{aligned}\right\} \quad \text{on } \Gamma. \tag{1.12}$$

For the exterior problem, the single and the double layer operator, taken on the boundary, read

$$\left.\begin{aligned}(\mathcal{V}_e u)(\boldsymbol{x}) &= \int_\Gamma \frac{u(\boldsymbol{y})}{4\pi\varepsilon\|\boldsymbol{x}-\boldsymbol{y}\|}\,\mathrm{d}\sigma_{\boldsymbol{y}} \\ (\mathcal{K}_e u)(\boldsymbol{x}) &= \int_\Gamma \frac{\langle \boldsymbol{n}(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|^3} u(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}}\end{aligned}\right\} \quad \text{on } \Gamma. \tag{1.13}$$

To treat the inhomogeneity, we require the Newton potential be defined in the entire space,

$$\mathcal{N}_\rho(\boldsymbol{x}) = \int_\Omega \frac{\rho(\boldsymbol{y})}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|}\,\mathrm{d}\boldsymbol{y} \qquad \text{in } \mathbb{R}^3. \tag{1.14}$$

It solves the partial differential equation

$$\begin{aligned}-\Delta\mathcal{N}_{\rho_i}(\boldsymbol{x}) &= \rho(\boldsymbol{x}) && \text{in } \Omega, \\ -\Delta\mathcal{N}_{\rho_e}(\boldsymbol{x}) &= 0 && \text{in } \Omega^c,\end{aligned} \tag{1.15}$$

with the transmission condition at the boundary

$$\mathcal{N}_{\rho_e}(\boldsymbol{x}) = \mathcal{N}_{\rho_i}(\boldsymbol{x}), \qquad \langle \nabla\mathcal{N}_{\rho_i}(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle = \langle \nabla\mathcal{N}_{\rho_e}(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle \qquad \text{on } \Gamma \tag{1.16}$$

and the decay condition

$$|\mathcal{N}_{\rho_e}(\boldsymbol{x})| = \mathcal{O}\big(\|\boldsymbol{x}\|^{-1}\big) \qquad \text{as } \|\boldsymbol{x}\| \to \infty.$$

As seen in [69], the Newton potential solves the following boundary integral equation

$$\left(\mathcal{V}_i \frac{\partial\mathcal{N}_\rho}{\partial\boldsymbol{n}}\right)(\boldsymbol{x}) = \left(\left(\mathcal{K}_i - \frac{1}{2}\right)\mathcal{N}_\rho\right)(\boldsymbol{x}) \qquad \text{on } \Gamma. \tag{1.17}$$

In particular, for the neutral solution case, we obviously have the following relation between the interior and exterior boundary operators from eq. (1.12) and eq. (1.13)

$$\varepsilon\mathcal{V}_e = \mathcal{V}_i \text{ and } \mathcal{K}_e = \mathcal{K}_i. \tag{1.18}$$

1.6.1.2. Ionic solutions

A small change in the neutral solution equations can lead to simulations for salt solutions. One has

$$\begin{aligned}-\Delta u_i(\boldsymbol{x}) &= \rho(\boldsymbol{x}) && \text{in } \Omega, \\ -\varepsilon\Delta u_e(\boldsymbol{x}) + \kappa^2 u_e(\boldsymbol{x}) &= 0 && \text{in } \Omega^c, \\ u_i(\boldsymbol{x}) = u_e(\boldsymbol{x}),\ \langle \nabla u_i(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle &= \langle \varepsilon\nabla u_e(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle && \text{on } \Gamma, \\ |u_e(\boldsymbol{x})| &= \mathcal{O}\big(\|\boldsymbol{x}\|^{-1}\big) && \text{as } \|\boldsymbol{x}\| \to \infty,\end{aligned} \tag{1.19}$$

where κ describes the ionic character of the solution and is inverse proportional to the Debye length.

The interior problem stays the same, the interior operators thus correspond to the ones found in eq. (1.12) and the Newton potential to the one in eq. (1.14). Whereas, the exterior single and double layer operators for $\boldsymbol{x} \in \Gamma$ now become:

$$
\begin{aligned}
(\mathcal{V}_e u)(\boldsymbol{x}) &= \int_\Gamma \frac{u(\boldsymbol{y}) e^{-\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{4\pi\varepsilon\|\boldsymbol{x}-\boldsymbol{y}\|}\, \mathrm{d}\sigma_{\boldsymbol{y}}, \\
(\mathcal{K}_e u)(\boldsymbol{x}) &= \int_\Gamma \frac{\langle \boldsymbol{n}(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle (1+\kappa\|\boldsymbol{x}-\boldsymbol{y}\|) e^{-\kappa\|\boldsymbol{x}-\boldsymbol{y}\|}}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|^3} u(\boldsymbol{y})\, \mathrm{d}\sigma_{\boldsymbol{y}}.
\end{aligned}
$$

The same system of boundary integral operators arises as seen in eq. (1.11) by inserting the appropriate operators. For more details about using the PCM for ionic solutions, see [6].

1.6.1.3. Liquid crystals

Liquid crystals are materials that have properties between the solid crystal and the liquid phase. They might have a preferred orientation, like crystals, but still flow, like liquids. Liquid crystals can be modelled using the PCM starting from the neutral solution case by replacing the dielectric constant by a tensor, $\boldsymbol{\varepsilon} \in \mathbb{R}^{3\times 3}$, containing the permittivity in each direction and solving the following system of differential equations:

$$
\begin{aligned}
-\Delta u_i(\boldsymbol{x}) &= \rho(\boldsymbol{x}) && \text{in } \Omega, \\
-\operatorname{div}(\boldsymbol{\varepsilon}\nabla u_e(\boldsymbol{x})) &= 0 && \text{in } \Omega^c, \\
u_i(\boldsymbol{x}) = u_e(\boldsymbol{x}),\ \langle \nabla u_i(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle &= \langle \boldsymbol{\varepsilon}\nabla u_e(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle && \text{on } \Gamma, \\
|u_e(\boldsymbol{x})| &= \mathcal{O}\big(\|\boldsymbol{x}\|^{-1}\big) && \text{as } \|\boldsymbol{x}\| \to \infty.
\end{aligned}
\tag{1.20}
$$

Again, a system of integral operators similar to eq. (1.11) has to be solved by using the corresponding operators and the jump conditions on the boundary from eq. (1.20). In the present case, the exterior boundary integral operators have the form

$$
\left.
\begin{aligned}
(\mathcal{V}_e u)(\boldsymbol{x}) &= \int_\Gamma \frac{u(\boldsymbol{y})}{4\pi\sqrt{\det(\boldsymbol{\varepsilon})}\|\boldsymbol{x}-\boldsymbol{y}\|_{\boldsymbol{\varepsilon}^{-1}}}\, \mathrm{d}\sigma_{\boldsymbol{y}}, \\
(\mathcal{K}_e u)(\boldsymbol{x}) &= \int_\Gamma \frac{\langle \boldsymbol{n}(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle}{4\pi\sqrt{\det(\boldsymbol{\varepsilon})}\|\boldsymbol{x}-\boldsymbol{y}\|^3_{\boldsymbol{\varepsilon}^{-1}}} u(\boldsymbol{y})\, \mathrm{d}\sigma_{\boldsymbol{y}},
\end{aligned}
\right\} \quad \text{on } \Gamma,
$$

with the norm $\|\boldsymbol{x}\|_{\boldsymbol{\varepsilon}^{-1}} := \sqrt{\langle \boldsymbol{x}, \boldsymbol{\varepsilon}^{-1}\boldsymbol{x}\rangle}$. The Dirichlet-to-Neumann maps in case the dielectric constant is a tensor become

$$
\begin{aligned}
\left(\mathcal{V}_i \langle \nabla u_i, \boldsymbol{n}\rangle\right)(\boldsymbol{x}) &= \left(\left(\mathcal{K}_i + \frac{1}{2}\right) u_i\right)(\boldsymbol{x}) - \mathcal{N}_\rho(\boldsymbol{x}), \\
\left(\mathcal{V}_e \langle \boldsymbol{\varepsilon}\nabla u_e, \boldsymbol{n}\rangle\right)(\boldsymbol{x}) &= \left(\left(\mathcal{K}_e - \frac{1}{2}\right) u_e\right)(\boldsymbol{x}).
\end{aligned}
\tag{1.21}
$$

More details about liquid crystals and the use of PCM can be found in [6].

1.6.2. The interaction energy

Having solved for the electrostatic potential, $u(\boldsymbol{x})$, the interaction energy between two charge distributions inside the cavity, $\rho(\boldsymbol{x})$ and $\rho'(\boldsymbol{x})$, can be calculated as

$$e(\rho, \rho') = \int_{\mathbb{R}^3} u(\boldsymbol{x})\rho'(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}. \tag{1.22}$$

Defining the apparent surface charge (ASC), $\sigma(\boldsymbol{x})$, as described in [7] by

$$\left(\mathcal{V}_i\sigma\right)(\boldsymbol{x}) = u(\boldsymbol{x}) - \mathcal{N}_\rho(\boldsymbol{x}) \qquad \text{on } \Gamma \tag{1.23}$$

and inserting this expression into the equation of the interaction energy, one gets the following splitting:

$$\begin{aligned} e(\rho, \rho') &= \int_{\mathbb{R}^3} u(\boldsymbol{x})\rho'(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\ &= \int_{\mathbb{R}^3} \mathcal{N}_\rho(\boldsymbol{x})\rho'(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int_{\mathbb{R}^3} \left(u_i(\boldsymbol{x}) - \mathcal{N}_\rho(\boldsymbol{x})\right)\rho'(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\ &= \int_{\mathbb{R}^3} \mathcal{N}_\rho(\boldsymbol{x})\rho'(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \int_{\mathbb{R}^3} \left(\mathcal{V}_i\sigma\right)(\boldsymbol{x})\rho'(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\ &= \int_{\mathbb{R}^3}\int_{\mathbb{R}^3} \frac{\rho(\boldsymbol{x})\rho'(\boldsymbol{y})}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|} \, \mathrm{d}\boldsymbol{y} \, \mathrm{d}\boldsymbol{x} + \int_{\mathbb{R}^3}\int_{\Gamma} \frac{\rho'(\boldsymbol{x})\sigma(\boldsymbol{y})}{4\pi\|\boldsymbol{x}-\boldsymbol{y}\|} \, \mathrm{d}\sigma_{\boldsymbol{y}} \, \mathrm{d}\boldsymbol{x}. \end{aligned}$$

Consequently, the interaction energy is represented as the sum of a vacuum term and a correction term which involves the ASC, depending only on the surface coordinate $\boldsymbol{y} \in \Gamma$.

1.6.3. The apparent surface charge and the boundary integral equation

As shown in [7], the ASC defined in eq. (1.23) is the unique solution to the following boundary integral equation, which is solved in the IEFPCM:

$$\left[\mathcal{V}_e\left(\mathcal{K}_i^T + \frac{1}{2}\right) - \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{V}_i\right]\sigma = \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{N}_\rho - \mathcal{V}_e\mathcal{V}_i^{-1}\left(\mathcal{K}_i - \frac{1}{2}\right)\mathcal{N}_\rho. \tag{1.24}$$

The interior and exterior operators depend on the problem under consideration.

For the derivation of eq. (1.24), one has to start from the general interior and exterior Dirichlet-to-Neumann maps found in eq. (1.21). The steps for the general model will be shown in the following and specific integral equations will be then computed for the neutral solution case.

1.6.3.1. Boundary integral equations for the general case

The starting point for the derivation of the ASC are the Dirichlet-to-Neumann maps from eq. (1.21). Using the transmission conditions $u_i(\boldsymbol{x}) = u_e(\boldsymbol{x})$ and $\langle \nabla u_i(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle = \langle \boldsymbol{\varepsilon}\nabla u_e(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle$, respectively, in the most general form found in eq. (1.20), one gets

to the system of equations only depending on the interior value of the electrostatic potential:

$$\begin{aligned}\left(\mathcal{V}_i\langle\nabla u_i, \boldsymbol{n}\rangle\right)(\boldsymbol{x}) &= \left(\left(\mathcal{K}_i + \frac{1}{2}\right) u_i\right)(\boldsymbol{x}) - \mathcal{N}_\rho(\boldsymbol{x}),\\ \left(\mathcal{V}_e\langle\nabla u_i, \boldsymbol{n}\rangle\right)(\boldsymbol{x}) &= \left(\left(\mathcal{K}_e - \frac{1}{2}\right) u_i\right)(\boldsymbol{x}).\end{aligned} \tag{1.25}$$

Since the exterior single layer operator $\mathcal{V}_e$ can in general not be transformed into the interior single layer operator $\mathcal{V}_i$, both equations need to be multiplied by the inverse of the respective single layer operators, which leads to

$$\begin{aligned}\langle\nabla u_i(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle &= \left(\mathcal{V}_i^{-1}\left(\mathcal{K}_i + \frac{1}{2}\right) u_i\right)(\boldsymbol{x}) - \left(\mathcal{V}_i^{-1}\mathcal{N}_\rho\right)(\boldsymbol{x}),\\ \langle\nabla u_i(\boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})\rangle &= \left(\mathcal{V}_e^{-1}\left(\mathcal{K}_e - \frac{1}{2}\right) u_i\right)(\boldsymbol{x}).\end{aligned} \tag{1.26}$$

By subtracting the second equation from the first one in the system of eq. (1.26), the electrostatic potential can be written as

$$u_i = \mathcal{B}^{-1}\mathcal{V}_i^{-1}\mathcal{N}_\rho, \tag{1.27}$$

in terms of the Newton potential. Thereby the operator $\mathcal{B}$ is given by

$$\mathcal{B} = \mathcal{V}_i^{-1}\left(\mathcal{K}_i + \frac{1}{2}\right) - \mathcal{V}_e^{-1}\left(\mathcal{K}_e - \frac{1}{2}\right).$$

Inserting eq. (1.27) into the definition of the ASC in eq. (1.23) and multiplying by the operator $\mathcal{B}$, one gets

$$\mathcal{B}\mathcal{V}_i\sigma = \mathcal{V}_i^{-1}\mathcal{N}_\rho - \mathcal{B}\mathcal{N}_\rho.$$

Multiplying this equation with $\mathcal{V}_e$ and using the commutation relation of the single layer and the double layer operators, $\mathcal{K}\mathcal{V} = \mathcal{V}\mathcal{K}^T$, [69], leads to the form of the equation used in practice

$$\left[\mathcal{V}_e\left(\mathcal{K}_i^T + \frac{1}{2}\right) - \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{V}_i\right]\sigma = \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{N}_\rho - \mathcal{V}_e\mathcal{V}_i^{-1}\left(\mathcal{K}_i - \frac{1}{2}\right)\mathcal{N}_\rho.$$

The Dirichlet-to-Neumann map for the Newton potential found in eq. (1.17) could be used to bring the equation in a different form,

$$\left[\mathcal{V}_e\left(\mathcal{K}_i^T + \frac{1}{2}\right) - \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{V}_i\right]\sigma = \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{N}_\rho - \mathcal{V}_e\frac{\partial\mathcal{N}_\rho}{\partial\boldsymbol{n}}, \tag{1.28}$$

which however is not used in practice since the ASC would depend also on the derivative of the Newton potential and not only on its value.

1.6.3.2. Boundary integral equations for the neutral solutions case

For the neutral solution case, the boundary integral eq. (1.27) can be modified further by using the operator equivalences in eq. (1.18). By inserting these properties into eq. (1.27), one arrives at

$$u_i = \frac{1}{\varepsilon - 1}\mathcal{A}^{-1}\mathcal{N}_\rho \qquad \text{on } \Gamma, \tag{1.29}$$

where the operator $\mathcal{A}$ is given by

$$\mathcal{A} = \frac{\varepsilon + 1}{2\,(\varepsilon - 1)} - \mathcal{K}_i. \tag{1.30}$$

Using the result found in eq. (1.29) in the definition of the ASC in eq. (1.23), one only has to solve a first kind integral equation to get the ASC. This equation only involves the Newton potential $\mathcal{N}_\rho$ as the right hand side:

$$\mathcal{V}_i\sigma = \left(\frac{1}{\varepsilon - 1}\mathcal{A}^{-1} - 1\right)\mathcal{N}_\rho \qquad \text{on } \Gamma. \tag{1.31}$$

By multiplying the first kind integral equation found in (1.31) by the operator $\mathcal{A}$, using the Dirichlet-to-Neumann map for the Newton potential from eq. (1.17) and again the commutation relation $\mathcal{K}\mathcal{V} = \mathcal{V}\mathcal{K}^T$, one gets the second kind integral equation

$$\mathcal{A}^T\sigma = \frac{\partial \mathcal{N}_\rho}{\partial \boldsymbol{n}} \qquad \text{on } \Gamma. \tag{1.32}$$

Nonetheless, due to the derivatives of the Newton potential appearing on the right, this boundary integral equation is rarely used in practice.

The solution of boundary integral equations will be treated in chpt. 2, in particular, the PDE solvers for the integral equation for PCM are discussed in sec. 2.5. In chpt. 3, an implementation using wavelets, which has the advantage of leading to sparse system matrices, will be described. The description of the molecular cavity from eq. (1.9), eq. (1.19) and eq. (1.20) will be discussed in chpt. 4.

CHAPTER 2

Solving boundary integral equations

Boundary integral equations, such as the IEFPCM eq. (1.24), can conveniently be solved numerically by the application of the boundary element method (BEM). Both, the integral operators and their associated function spaces, are defined solely on the boundary of the cavity. The discretization of the system can be carried out by using either the *collocation* or the *Galerkin method*, [23]. In order to compute the Galerkin solution of the boundary integral equation, one introduces the variational formulation, also called the weak form, of the integral equation at hand.

In this chapter, a general boundary integral equation, $(\mathcal{A}u)(\boldsymbol{x}) = \rho(\boldsymbol{x})$, with a boundary integral operator $\mathcal{A} : H^q(\Gamma) \to H^{-q}(\Gamma)$ of order $2q$,

$$(\mathcal{A}u)(\boldsymbol{x}) = \int_\Gamma k(\boldsymbol{x}, \boldsymbol{y}) u(\boldsymbol{y}) \, \mathrm{d}\sigma_{\boldsymbol{y}}, \tag{2.1}$$

and the right hand side $\rho \in H^{-q}(\Gamma)$ will be considered. The solution hence satisfies $u \in H^q(\Gamma)$.

2.1. Function spaces

In order to understand the mathematical background behind the boundary integral equations and the solvers used to find their solution, one has to first take a look at the function spaces involved in the equation and their corresponding norms, as introduced for example in [69].

2.1.1. C^k function spaces

A multiindex is a vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}_0^n$. Its modulus $|\boldsymbol{\alpha}|$ is defined as $|\boldsymbol{\alpha}| := \sum_{i=1}^n \alpha_i$. Multiindices can be used as a short hand notation for the derivatives of higher order of a given function $u : \mathbb{R}^n \to \mathbb{R}$, i.e.

$$\partial^{\boldsymbol{\alpha}} u(\boldsymbol{x}) = \left(\frac{\partial}{\partial x_1}\right)^{\alpha_1} \left(\frac{\partial}{\partial x_2}\right)^{\alpha_2} \ldots \left(\frac{\partial}{\partial x_n}\right)^{\alpha_n} u(\boldsymbol{x}),$$

where $n \in \mathbb{N}$ is the dimension of the space under consideration.

Recall that a vector space equipped with a norm is called a normed vector space. In case the vector space is complete, it is called a Banach space. If the norm is induced by a scalar product, the space is called a Hilbert space.

Definition 2.2. *Let $\Omega \subset \mathbb{R}^n$ be a bounded domain and $k \in \mathbb{N}$. The Banach space $C^k(\Omega)$ is defined as the space of all functions defined on Ω that are k times continuously differentiable and are bounded in the norm*

$$\|u\|_{C^k(\Omega)} = \sum_{|\boldsymbol{\alpha}| \leq k} \sup_{\boldsymbol{x} \in \Omega} |\partial^{\boldsymbol{\alpha}} u(\boldsymbol{x})|.$$

The special case of infinitely differentiable functions with compact support is denoted by

$$C_0^\infty(\Omega) := \{u \in C^\infty(\Omega) : \ \operatorname{supp} u \subset \Omega\}.$$

The Banach space of Hölder continuous functions $C^{k,\kappa}(\Omega)$, where $k \in \mathbb{N}$ and $\kappa \in (0, 1]$, is equipped with the norm

$$\|u\|_{C^{k,\kappa}(\Omega)} := \|u\|_{C^k(\Omega)} + \sum_{|\boldsymbol{\alpha}|=k} \sup_{\boldsymbol{x},\boldsymbol{y} \in \Omega, \ \boldsymbol{x} \neq \boldsymbol{y}} \frac{|\partial^{\boldsymbol{\alpha}} u(\boldsymbol{x}) - \partial^{\boldsymbol{\alpha}} u(\boldsymbol{y})|}{\|\boldsymbol{x} - \boldsymbol{y}\|^\kappa}.$$

The elements of this function space are functions $f \in C^k(\Omega)$ for which all k-th derivatives satisfy the Hölder condition

$$|\partial^{\boldsymbol{\alpha}} f(\boldsymbol{x}) - \partial^{\boldsymbol{\alpha}} f(\boldsymbol{y})| \leq C \|\boldsymbol{x} - \boldsymbol{y}\|^\kappa, \qquad |\boldsymbol{\alpha}| = k.$$

2.1.2. L^p spaces and their duals

Definition 2.3 (L^p spaces)**.** *Let Ω be a domain in $\mathbb{R}^n$ and $1 \leq p \leq \infty$. The Lebesgue function spaces $L^p(\Omega)$ are the Banach spaces of measurable functions defined on Ω, where the associated norm*

$$\|u\|_{L^p(\Omega)} := \begin{cases} \left(\int_\Omega |u(\boldsymbol{x})|^p \, \mathrm{d}\boldsymbol{x} \right)^{1/p}, & 1 \leq p < \infty, \\ \operatorname*{ess\,sup}_{\boldsymbol{x} \in \Omega} \{|u(\boldsymbol{x})|\}, & p = \infty, \end{cases}$$

is bounded.

Two functions of $L^p(\Omega)$ are said to be equal if they are distinct only on a domain of measure zero. Note that for $p = 2$ one gets a Hilbert space, namely the space of quadratically integrable functions, $L^2(\Omega)$, with the scalar product defined as

$$\langle u, v \rangle_{L^2(\Omega)} := \int_\Omega u(\boldsymbol{x}) v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}.$$

Additionally, $L^2(\Gamma)$ is the Hilbert space of quadratically integrable functions which are bounded with respect to the scalar product taken on the boundary of the domain, $\Gamma = \partial\Omega$,

$$\langle u, v \rangle_{L^2(\Gamma)} := \int_\Gamma u(\boldsymbol{x}) v(\boldsymbol{x}) \, \mathrm{d}\sigma_{\boldsymbol{x}}.$$

2.1.3. Weak derivatives and Sobolev spaces on domains

Definition 2.4 (Weak derivatives)**.** *Let Ω be a bounded domain in $\mathbb{R}^n$ with boundary Γ. The function $u \in L^1(\Omega)$ has a weak derivative if a function $v \in L^1(\Omega)$ exists such that*

$$\langle v, \varphi \rangle_{L^2(\Omega)} = -\left\langle u, \frac{\partial \varphi}{\partial x_i} \right\rangle_{L^2(\Omega)} \qquad \text{for all } \varphi \in C_0^\infty(\Omega).$$

The weak derivative is denoted by $v = {}^{\partial u}/_{\partial x_i}$ and, in case u is differentiable, it coincides with the strong derivative.

Recursively, higher order and multivariate derivatives can be defined.

Definition 2.5 (Sobolev spaces $H^m(\Omega)$, $m \in \mathbb{N}$)**.** *The space of all functions in $L^2(\Omega)$, which are weakly differentiable of order $\boldsymbol{\alpha}$ up to $|\boldsymbol{\alpha}| \leq m \in \mathbb{N}$ and with the derivatives in $L^2(\Omega)$, is called the Sobolev space $H^m(\Omega)$. This space is a Hilbert space under consideration of the scalar product*

$$\langle u, v \rangle_{H^m(\Omega)} = \sum_{|\boldsymbol{\alpha}| \leq m} \langle \partial^{\boldsymbol{\alpha}} u, \partial^{\boldsymbol{\alpha}} v \rangle_{L^2(\Omega)}$$

and the induced norm

$$\|u\|_{H^m(\Omega)} = \sqrt{\langle u, u \rangle_{H^m(\Omega)}} = \sqrt{\sum_{|\boldsymbol{\alpha}| \leq m} \|\partial^{\boldsymbol{\alpha}} u\|^2_{L^2(\Omega)}}.$$

The extension of the Sobolev spaces to arbitrary $s \in \mathbb{R}$ can be done by extending the scalar product and the norms first to $0 < s \in \mathbb{R}$.

Definition 2.6 (Sobolev-Slobodeckij spaces $H^s(\Omega),\ 0 < s \in \mathbb{R}$)**.** *For $0 < s \in \mathbb{R}$ with $s = m + \beta$, $m \in \mathbb{N}$ and $\beta \in (0,1)$, the Hilbert space $H^s(\Omega)$ contains all functions $u \in H^m(\Omega)$ with bounded norm*

$$\|u\|_{H^s(\Omega)} = \sqrt{\langle u, u\rangle_{H^s(\Omega)}}$$

and the scalar product given by

$$\langle u, v\rangle_{H^s(\Omega)} = \langle u, v\rangle_{H^m(\Omega)} + \sum_{|\boldsymbol{\alpha}|=m} \int_\Omega \int_\Omega \frac{(\partial^{\boldsymbol{\alpha}} u(\boldsymbol{x}) - \partial^{\boldsymbol{\alpha}} u(\boldsymbol{y}))(\partial^{\boldsymbol{\alpha}} v(\boldsymbol{x}) - \partial^{\boldsymbol{\alpha}} v(\boldsymbol{y}))}{\|\boldsymbol{x} - \boldsymbol{y}\|^{n+2\beta}} \,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y}.$$

In the case of the equations (1.24), (1.28), (1.31), or (1.32), since $\Omega \in \mathbb{R}^3$, the parameter n is equal to three.

Next, let the space $H_0^s(\Omega)$ be defined by the completion of $C_0^\infty(\Omega)$ with respect to the previously defined norm, $\|\cdot\|_{H^s(\Omega)}$, i.e.

$$H_0^s(\Omega) := \overline{C_0^\infty(\Omega)}^{\|\cdot\|_{H^s(\Omega)}}.$$

Definition 2.7 (Sobolev-Slobodeckij spaces $H^{-s}(\Omega),\ 0 < s \in \mathbb{R}$)**.** *The dual space $H^{-s}(\Omega)$ of $H_0^s(\Omega)$ is defined as the space of linear maps from $H_0^s(\Omega)$ to $\mathbb{R}$, equipped with the dual norm*

$$\|u\|_{H^{-s}(\Omega)} = \sup_{0 \neq v \in H^s(\Omega)} \frac{\langle u, v\rangle_{L^2(\Omega)}}{\|v\|_{H^s(\Omega)}}. \tag{2.8}$$

2.1.4. Weak derivatives and Sobolev spaces on manifolds

Definition 2.9 ($N^{k,\kappa}$ property)**.** *The domain $\Omega \subset \mathbb{R}^n$ is said to possess the $N^{k,\kappa}$ property if for each point $\boldsymbol{x} \in \Gamma := \partial\Omega$ a neighbourhood, $U_{\boldsymbol{x}}$, an orthogonal coordinate transform, $\boldsymbol{A}_{\boldsymbol{x}} : \mathbb{R}^n \to \mathbb{R}^n$, two numbers, $\alpha_{\boldsymbol{x}}, \beta_{\boldsymbol{x}} > 0$, and a function, $a_{\boldsymbol{x}} : \mathbb{R}^{n-1} \to \mathbb{R}$ can be found such that the following properties hold.*

Let $y_1, y_2, \ldots, y_n$ be the new Cartesian coordinates given by the transform $\boldsymbol{A}_{\boldsymbol{x}}$, and let $W^{n-1}(\alpha_{\boldsymbol{x}})$ be the cube $\{y_1, y_2, \ldots, y_{n-1} : |y_i| < \alpha_{\boldsymbol{x}},\ i = 1, 2, \ldots, n-1\} \in \mathbb{R}^{n-1}$. Then:

- $a_{\boldsymbol{x}} \in C^{k,\kappa}(W^{n-1}(\alpha_{\boldsymbol{x}}))$.
- *The boundary $U_{\boldsymbol{x}} \cap \Gamma$ can be described by means of the new coordinates and the function $a_{\boldsymbol{x}}(y_1, y_2, \ldots, y_{n-1}) = y_n$, which leads to*

$$U_{\boldsymbol{x}} \cap \Omega = \left\{(y_1, \ldots, y_n) : (y_1, \ldots, y_{n-1}) \in W^{n-1}(\alpha_{\boldsymbol{x}}),\ y_n = a_{\boldsymbol{x}}(y_1, \ldots, y_{n-1})\right\}.$$

- *The displacement of $U_{\boldsymbol{x}} \cap \Gamma$ by $\beta_{\boldsymbol{x}}$ results in a surface that is inside of Ω*

$$\begin{aligned} U_{\boldsymbol{x}} \cap \Omega = \{&(y_1, \ldots, y_n) : |y_i| < \alpha_{\boldsymbol{x}},\ i = 1, \ldots, n-1, \\ &a_{\boldsymbol{x}}(y_1, \ldots, y_{n-1}) < y_n < a_{\boldsymbol{x}}(y_1, \ldots, y_{n-1}) + \beta_{\boldsymbol{x}}\} = V_{\boldsymbol{x}}^+. \end{aligned}$$

- *The displacement of $U_{\boldsymbol{x}} \cap \Gamma$ by $-\beta_{\boldsymbol{x}}$ results in a surface that is outside of Ω*

$$U_{\boldsymbol{x}} \cap \overline{\Omega}^c = \{(y_1, \ldots, y_n) : |y_i| < \alpha_{\boldsymbol{x}},\ i = 1, \ldots, n-1,\\ a_{\boldsymbol{x}}(y_1, \ldots, y_{n-1}) - \beta_{\boldsymbol{x}} < y_n < a_{\boldsymbol{x}}(y_1, \ldots, y_{n-1})\} = V_{\boldsymbol{x}}^-.$$

The number $\beta_{\boldsymbol{x}}$ represents the local inward normal of the surface. The third and fourth requirement ensure that Ω lies on one side of the surface Γ, meaning that Γ is indeed the boundary of the domain Ω. Arbitrary polyhedral regions have the $N^{0,1}$ property. Moreover, it can be shown that the $N^{k,\kappa}$ property is equivalent to $C^{k,\kappa}(\Omega)$, see [75] for details.

Theorem 2.10 (Uniform cone property)**.** *Bounded domains Ω that possess the $N^{0,1}$ property also have the uniform cone property.*

Definition 2.11 (Sobolev space $H^s(\Gamma),\ 0 < s \in \mathbb{R}$)**.** *If $\Omega \subset \mathbb{R}^n$ is bounded and has the $N^{k,\kappa}$ property, any function u defined on Γ can be decomposed into regions. Such a finite decomposition always exists and is obtained by a suitable partition of unity. Let $u(\boldsymbol{x}) = \sum_i^n u_i(\boldsymbol{x})$ with $\operatorname{supp} u_i \subset U_{\boldsymbol{x}}$, where the coordinate neighbourhoods, $U_{\boldsymbol{x}}$, have the properties found in def. 2.9. Then the norm of $\|u\|^2_{H^s(\Gamma)}$ is given with respect to the local functions $\|u\|^2_{H^s(\Gamma)} = \sum_i^n \|u_i\|^2_{H^s(\Gamma)}$ with the norms of the localized functions*

$$\|u_i\|^2_{H^s(\Gamma)} = \sum_{|\boldsymbol{\alpha}| \le s} \int_\Gamma |\partial^{\boldsymbol{\alpha}} u_i|^2 \,\mathrm{d}\sigma_{\boldsymbol{x}} + \sum_{|\boldsymbol{\alpha}| \le s} \int_\Gamma \int_\Gamma \frac{|\partial^{\boldsymbol{\alpha}} u_i(\boldsymbol{x}) - \partial^{\boldsymbol{\alpha}} u_i(\boldsymbol{y})|^2}{|\boldsymbol{x} - \boldsymbol{y}|^{n-1+2\beta}} \,\mathrm{d}\sigma_{\boldsymbol{x}} \,\mathrm{d}\sigma_{\boldsymbol{y}},$$

where $0 < s \in \mathbb{R}$ with $s = m + \beta$, $m \in \mathbb{N}$ and $\beta \in (0, 1)$. The partial derivatives are taken with respect to the coordinates $(y_1, \ldots, y_{n-1})$, which correspond to the coordinates of the local mappings $y_n = a_{\boldsymbol{x}}(y_1, \ldots, y_{n-1})$, as seen in def. 2.9, and the surface element is given by

$$\mathrm{d}\sigma_{\boldsymbol{x}} = \left[1 + \sum_{j=1}^{n-1} \left(\frac{\partial a_{\boldsymbol{x}}}{\partial y_j}\right)\right] \mathrm{d}y_1 \ldots \mathrm{d}y_{n-1}.$$

All functions from $L^2(\Gamma)$, whose $H^s(\Gamma)$-norm is bounded, belong to the Sobolev space $H^s(\Gamma)$.

Definition 2.12 (Interior trace operator)**.** *The interior trace is the continuous linear map*

$$\gamma_0^{int} : C(\overline{\Omega}) \to C(\Gamma)$$

with the property

$$\gamma_0^{int} f(\boldsymbol{x}) := \lim_{\boldsymbol{y} \in \Omega,\ \boldsymbol{y} \to \boldsymbol{x}} f(\boldsymbol{y})$$

for all $\boldsymbol{x} \in \Gamma$ and for all $f \in C(\overline{\Omega})$.

Lemma 2.13 (Continuation of the trace)**.** *For $\Omega \subset \mathbb{R}^n$ with a $C^{k,\kappa}$-boundary and $1/2 < s \le k + \kappa$, the interior trace can be extended to $H^s(\Omega)$ such that it is a unique linear map*

$$\gamma_0^{int} : H^s(\Omega) \to H^{s-1/2}(\Gamma)$$

with the property

$$\gamma_0^{int} f(\boldsymbol{x}) = f(\boldsymbol{x})\big|_{\boldsymbol{x}\in\Gamma}$$

for all $f \in C^{\lceil s \rceil}(\overline{\Omega})$.

In particular, the restriction to the boundary of the single and double layer operators from eq. (1.24) are well defined. For more details, see [75] and the references therein.

Lemma 2.14. *Let* Ω *be a Lipschitz domain with boundary* Γ. *Then, the boundary integral operators have the mapping properties*

$$\begin{aligned}
\mathcal{V} &: H^{-1/2+s}(\Gamma) \to H^{1/2+s}(\Gamma),\\
\mathcal{K} &: H^{1/2+s}(\Gamma) \to H^{1/2+s}(\Gamma),\\
\mathcal{K}^T &: H^{-1/2+s}(\Gamma) \to H^{-1/2+s}(\Gamma),
\end{aligned}$$

for all $s \in [-1/2, 1/2]$. *Moreover, the trace of the Newton potential satisfies*

$$\gamma_0^{int}\mathcal{N}_\rho : \tilde{H}^{-1+s}(\Omega) \to H^{1/2+s}(\Gamma), \qquad s \in [-1/2, 1/2]\,,$$

where $\tilde{H}^{-1+s}(\Omega)$ *denotes the dual of* $H^{1-s}(\Omega)$.

Note that a proof of this lemma is found in [11]. For the case of domains with higher regularity, the range of s can be extended correspondingly, see [69] for details.

Using the information of Lemma 2.14, the spaces for the boundary integral operators, found in chpt. 1, can be defined.

Corollary 2.15. *Recall the boundary integral eq.* (1.24)*:*

$$\left[\mathcal{V}_e\left(\mathcal{K}_i^T + \frac{1}{2}\right) - \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{V}_i\right]\sigma = \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{N}_\rho - \mathcal{V}_e\mathcal{V}_i^{-1}\left(\mathcal{K}_i - \frac{1}{2}\right)\mathcal{N}_\rho.$$

This equation was transformed into eq. (1.28)*:*

$$\left[\mathcal{V}_e\left(\mathcal{K}_i^T + \frac{1}{2}\right) - \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{V}_i\right]\sigma = \left(\mathcal{K}_e - \frac{1}{2}\right)\mathcal{N}_\rho - \mathcal{V}_e\frac{\partial\mathcal{N}_\rho}{\partial\boldsymbol{n}}.$$

The boundary integral operators on the left hand side of both integral equations are composed of interior and exterior single layer operators and double layer operators, which results in an operator which maps functions from $H^{-1/2}(\Gamma)$ *onto* $H^{1/2}(\Gamma)$*, i.e. it is an operator of order* $2q = -1$.

For the neutral solution case, eq. (1.31)*, it holds*

$$\mathcal{V}_i\sigma = \left(\frac{1}{\varepsilon - 1}\mathcal{A}^{-1} - 1\right)\mathcal{N}_\rho \qquad \text{on } \Gamma$$

with $\mathcal{A}$ *given by eq.* (1.30). *This is a Fredholm boundary integral equation of the first kind for the single layer operator,* $\mathcal{V}_i : H^{-1/2}(\Gamma) \to H^{1/2}(\Gamma)$*, which is again an operator of order* $2q = -1$.

The last equation has been changed further into eq. (1.32)*,*

$$\mathcal{A}^T\sigma = \frac{\partial \mathcal{N}_\rho}{\partial \boldsymbol{n}} \qquad \text{on } \Gamma,$$

making use of only the transposed of the double layer operator. This is a Fredholm boundary integral equation of second kind which involves an operator of order $2q = 0$*.*

2.2. Discretization of the solution

In order to be able to solve the boundary integral equation that arises, well chosen finite dimensional ansatz spaces are used. We are thus looking for the approximation of the solution $u(\boldsymbol{x})$ by a function $u_n(\boldsymbol{x})$ in the finite dimensional ansatz space

$$V_n = \operatorname{span}\{\varphi_0, \varphi_1, \ldots, \varphi_n\} \tag{2.16}$$

as follows

$$u_n(\boldsymbol{x}) = \sum_{i=0}^{n} \alpha_i \varphi_i(\boldsymbol{x}) = \langle \boldsymbol{u}_n, \boldsymbol{\Phi}_n(\boldsymbol{x}) \rangle, \tag{2.17}$$

where

$$\left.\begin{aligned} (\boldsymbol{u}_n)_i &= \alpha_i \\ (\left(\boldsymbol{\Phi}_n\right)(\boldsymbol{x}))_i &= \varphi_i(\boldsymbol{x}) \end{aligned}\right\} \qquad \text{for } i = 0, \ldots, n.$$

A common choice of ansatz functions results from introducing a partition of the surface into simple elements, typically triangles or squares, and defining piecewise polynomial ansatz functions on these finite elements.

2.2.1. Finite elements on the boundary

The boundary of the domain is partitioned in suitable finite subsets, called elements

$$\Gamma = \bigcup_{i=0}^{n} \Gamma_i, \tag{2.18}$$

where the ansatz functions, φ_i, will be defined on. In practice, a partitioning in simple geometric figures, for example triangles or squares, is encountered. By this partitioning, a mesh size, h, of the discretization of the surface is implicitly introduced. It is given by the radius, $h_i = {}^{\operatorname{diam}(\Gamma_i)}/_2$, of the circumscribed circle for each element $\Gamma_i \in \Gamma$. The mesh is called shape regular if there exists a number $\kappa > 0$ such that the elements contain a circle of radius ρ_i with

$$\rho_i \geq \frac{h_i}{\kappa}.$$

Additionally, if the radius of the circle can be chosen independently of the current element of the decomposition, but dependent on $h := \max h_i$, the decomposition is called quasi-uniform and fulfills the equation

$$\rho_i \geq \frac{h}{\kappa}.$$

The mesh size will have an influence when discussing the convergence of the boundary element method, but first the typical ansatz functions need to be introduced. For simplicity, the figures will be depicted for triangular meshes.

Definition 2.19 (Conforming mesh)**.** *A partition of the surface is called conforming if the intersection of two different elements is either a single point, called vertex, an entire edge or the empty set.*

An example of a non-conforming mesh is seen in fig. 2.20a as well as an example of a conforming mesh in fig. 2.20b. The meshes used in this thesis are all conforming meshes.

(a) Non-conforming mesh example. **(b)** Conforming mesh example.

Figure 2.20. Examples of a non-conforming and a conforming mesh.

2.2.1.1. Piecewise constant ansatz functions

The simplest ansatz space is the space of functions which are constant on each element. For a given partition of the surface, as in eq. (2.18), the basis of *piecewise constant ansatz functions* is defined as follows:

$$\varphi_i(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in \Gamma_i, \\ 0, & \boldsymbol{x} \notin \Gamma_i. \end{cases}$$

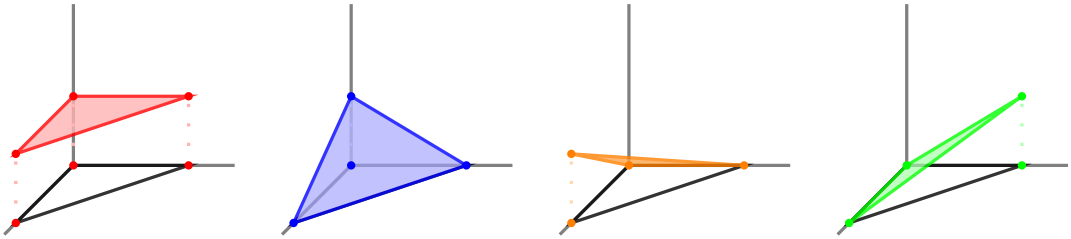
A schematic of a piecewise constant ansatz function can be found in fig. 2.21a.

Discretizing the surface in smaller elements would result in a smaller value for the mesh size, h, which in turn influences the approximation to the exact solution u, as will be shown later.

2.2.1.2. Piecewise linear ansatz functions

Another basis of the discrete space, V_n, is to use the space of functions that are linear polynomials on each element and globally continuous. The ansatz function $\varphi_i \in V_n$ is uniquely defined by the values at the vertices of the elements, which satisfy

$$\varphi_i(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} = \boldsymbol{p}_i, \\ 0, & \boldsymbol{x} = \boldsymbol{p}_j,\ i \neq j, \end{cases}$$



(a) Piecewise constant ansatz function.

(b) First linear ansatz function, value 1 in the first vertex.

(c) Second linear ansatz function, value 1 in the second vertex.

(d) Third linear ansatz function, value 1 in the third vertex.

Figure 2.21. Piecewise constant and piecewise linear ansatz functions, depicted for a triangle.

where $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are denoting vertices of the surface mesh. A schematic of piecewise bilinear ansatz functions which have non-zero contributions on an element can be found in fig. 2.21b, fig. 2.21c and fig. 2.21d.

The Euler formula gives a relation between the number of vertices, n_v, the number of elements, n_Δ, and the number of edges, n_e,

$$n_v + n_\Delta - n_e = 2.$$

This holds on any closed surface, which is topologically equivalent to a sphere, as is the case of simple molecules. If the molecules have a tunnel, like a doughnut, the equation becomes

$$n_v + n_\Delta - n_e = 0,$$

and, for any further hole, the number on the right hand side would decrease by 2. This means that the number of vertices is proportional to the number of elements. In particular, it implies that both, the number of piecewise linear ansatz functions and the number of piecewise constant ansatz functions, grow asymptotically linear with the number of elements.

2.3. Collocation method

The first method that comes to mind in order to solve the discrete version of eq. (2.1) is the so-called collocation method, which requires that the values of the discrete system coincide in a set of $n+1$ disjoint sampling points,

$$K = \{\boldsymbol{\xi}_0, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_n\} \subset \Gamma,$$

also called collocation points. This leads to solving the following equations:

$$(\mathcal{A}u_n)(\boldsymbol{\xi}_i) = \rho(\boldsymbol{\xi}_i) \qquad \text{for all } i = 0, \ldots, n.$$

By assembling the system of linear equations for all values of $\boldsymbol{\xi}_i$, one gets

$$\boldsymbol{M}_n \boldsymbol{u}_n = \boldsymbol{\rho}_n \tag{2.22}$$

with the matrix entries

$$(\boldsymbol{M}_n)_{i,j} = (\mathcal{A}\varphi_j)(\boldsymbol{\xi}_i) = \int_\Gamma k(\boldsymbol{\xi}_i, \boldsymbol{y})\varphi_j(\boldsymbol{y})\,\mathrm{d}\sigma_{\boldsymbol{y}},$$

the right hand side

$$(\boldsymbol{\rho}_n)_i = \rho(\boldsymbol{\xi}_i),$$

and the coefficient vector $(\boldsymbol{u}_n)_i = \alpha_i$ as given in eq. (2.17).

The approximation $u_n(\boldsymbol{x})$ is uniquely defined if the associated matrix $\boldsymbol{M}_n$ is regular. This means that the collocation points have to be chosen in such a way that the coefficients α_i of $u_n(\boldsymbol{x})$ are uniquely defined. The optimal number of collocation points for a given discretization of the solution has exactly $n+1$ points, that is equal to the number of ansatz functions used. This results in solving a system of linear equations with a square matrix $\boldsymbol{M}_n$ which is in general dense, see [23] for more details. A clever choice of collocation points would be the midpoints of the elements for piecewise constant ansatz functions or the mesh vertices for piecewise linear ansatz functions.

The convergence of the solution of the discrete linear system (2.22) depends on the mesh size implicitly defined by n. We shall restrict ourselves to an integral operator of order zero and a quasi-uniform mesh. Then, according to [25], given n elements, the discrete solution of the collocation method, u_n, converges towards the exact solution, u, with respect to the mesh size with the rate

$$\|u - u_n\|_{L^\infty(\Omega)} \leq Ch^d.$$

Here, d denotes the approximation order of the ansatz functions used, that is $d = 1$ for the piecewise constant ansatz functions and $d = 2$ for the piecewise linear ansatz functions. For generalizations to integral operators of non-zero order, see for example [25, 42] and the references therein.

2.4. Galerkin method

The introduction of the Galerkin method requires the variational formulation or weak formulation of the boundary integral equation. The variational formulation of a general boundary integral eq. (2.1) arises by multiplying the equation with a test function and integrating over the entire domain. This results in solving the problem:

$$\begin{aligned} &\text{Seek } u \in H^q(\Gamma) \text{ such that} \\ &\qquad \langle \mathcal{A}u, v\rangle_{L^2(\Gamma)} = \langle \rho, v\rangle_{L^2(\Gamma)} \qquad \text{for all } v \in H^{-q}(\Gamma). \end{aligned} \tag{2.23}$$

It is obvious that the solution, $u(\boldsymbol{x})$, of eq. (2.1) also satisfies the variational formulation (2.23). For the reverse direction, the norm definition found in (2.8) can be used. In particular, if $\tilde{u}(\boldsymbol{x})$ would be the solution of the variational formulation, one has

$$\langle \mathcal{A}\tilde{u} - \rho, v\rangle_{L^2(\Gamma)} = 0 \qquad \text{for all } v \in H^q(\Gamma).$$

This means that $0 = \mathcal{A}\tilde{u} - \rho \in H^{-q}(\Gamma)$, which in turn implies that $\tilde{u} \in H^q(\Gamma)$ has to be a solution to the integral eq. (2.1), cf. [69].

For the Galerkin method, the infinite dimensional energy space $H^{-q}(\Gamma)$ is replaced by a finite dimensional trial space W_n. In our case, the trial space is chosen equal to the ansatz space, meaning that $W_n = V_n$:

$$\begin{aligned} &\text{Seek } u_n \in V_n \text{ such that} \\ &\quad \langle \mathcal{A}u_n, v_n \rangle_{L^2(\Gamma)} = \langle \rho, v_n \rangle_{L^2(\Gamma)} \qquad \text{for all } v_n \in V_n. \end{aligned}$$

We shall use the same trial space as in (2.16).

The problem can be simplified even more, if one takes a look at the basis that spans the space $V_n = \operatorname{span}\{\varphi_0, \varphi_1, \ldots, \varphi_n\}$. Given such a basis, it suffices to test with each of the function of the basis, i.e.

$$\langle \mathcal{A}u_n, \varphi_i \rangle_{L^2(\Gamma)} = \langle \rho, \varphi_i \rangle_{L^2(\Gamma)}, \qquad i = 0, \ldots, n.$$

Inserting (2.17) into the previous equation and using the linearity of the integral operator under consideration leads to

$$\sum_{i=0}^{n} \alpha_i \langle \mathcal{A}\varphi_i, \varphi_j \rangle_{L^2(\Gamma)} = \langle \rho, \varphi_j \rangle_{L^2(\Gamma)}, \qquad j = 0, \ldots, n.$$

This is equivalent to the system of linear equations

$$\boldsymbol{M}_n \boldsymbol{u}_n = \boldsymbol{\rho}_n$$

with the matrix entries

$$(\boldsymbol{M}_n)_{i,j} = \langle \mathcal{A}\varphi_j, \varphi_i \rangle_{L^2(\Gamma)},$$

the right hand side

$$(\boldsymbol{\rho}_n)_i = \langle \rho, \varphi_i \rangle_{L^2(\Gamma)},$$

and the coefficient vector $(\boldsymbol{u}_n)_i = \alpha_i$ as given in (2.17).

By using the Céa Lemma and the approximation property of the ansatz spaces, the following convergence result can be proven for the approximation, $u_n \in V_n$. The difference to the exact solution, $u \in H^q(\Omega)$, in the energy norm of the operator at hand is given by

$$\|u - u_n\|_{H^q(\Gamma)} \leq C \inf_{v_n \in V_n(\Gamma)} \|u - v_n\|_{H^q(\Gamma)} \leq C h^{d-q} \|u\|_{H^d(\Gamma)},$$

provided that u is sufficiently regular and the finite element mesh is quasi-uniform and where d is the approximation order of the ansatz functions used. This means that, given a high enough resolution of the space V_n, which implicitly leads to the mesh size $h \to 0$ for fixed d, one can approximate the solution u of the boundary integral eq. (2.1) with arbitrary precision.

Additionally, the Aubin-Nitsche trick can double the convergenece rate, leading to the error estimate

$$\|u - u_n\|_{H^{2q-d}(\Gamma)} \leq C h^{2(d-q)} \|u\|_{H^d(\Gamma)}.$$

For more details concerning the Galerkin method, see e.g. [69].

2.5. Computational chemistry packages

A brief overview of software packages and their capabilities will shed some light on the current state of the art. This summary is by no means complete, for more details please consult the review [73] and the references therein.

2.5.1. PDE solvers

There are several quantum chemical packages out there that tackle the PCM equations. All of these implementations use the collocation method, differing sometimes in the basis functions used. A more exotic choice can be found in the continuous surface charge (CSC) and the switching Gaussians (SWIG) methods which use spherical Gaussians as basis functions, [43, 44, 64]. These implementations can be found in the commercial codes Gaussian and Q-Chem.

The fast multipole method, introduced by Greengard and Rokhlin in [20, 21, 22, 61], was extended to the PCM by Kirkwood and Onsager, [38, 39, 49]. Here, the potential inside the cavity is expressed by an expansion in terms of spherical harmonics such that fast summation techniques can be applied. This method has been integrated into the Gaussian and Dalton packages.

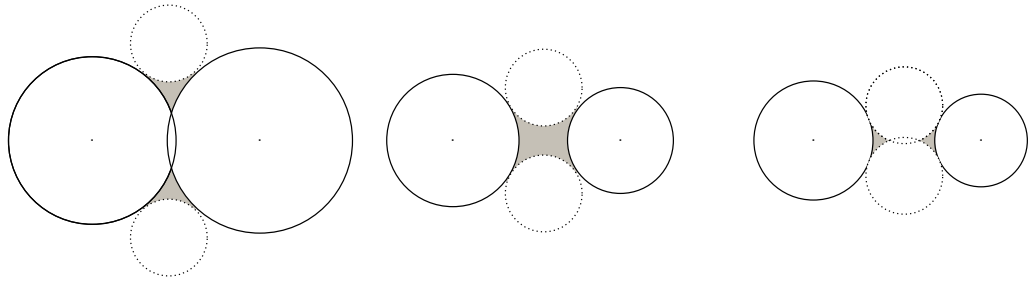
The only implementation of a Galerkin approach to the solution of the boundary integral equations is the one described in [5, 28, 29, 74] and is the foundation stone of this thesis. It was incorporated in the template based module PCMSolver which is included in several packages, like Psi4, [50], LSDalton, [1], Dirac, [59], and ReSpect, [60].

2.5.2. Existing SES cavity generation algorithms in chemistry packages

Due to the fact that a bad grid on the cavity can break the performance of boundary element method codes and a poor precision of the cavity leads to a poor precision of the measurable quantities calculated, a lot of effort has to be spent in the generation of the cavities and their discretization. Since this thesis is mainly concerned with the mesh generation for the Connolly surface, an overview of cavity generators for this type of cavity is given here.

2.5.2.1. GEPOL

The GEPOL algorithm has been proposed by Pascual-Ahuir and Silla in [51, 52, 67] for approximating the SES by convex spherical partial surfaces. This is done by inserting additional spheres, called ghost atoms, in the space which is inaccessible to the fluid. Three types of cases are considered: the case of overlapping spheres, which arise only in the case of toroidal partial surfaces, and two cases of non-overlapping spheres. The non-overlapping cases include the generation of one partial surface, for the toroidal case, or two partial surfaces. The three cases for the interaction between two spheres are depicted in fig. 2.24.



(a) Generation of one surface from the interaction of overlapping spheres.

(b) Generation of one surface from the interaction of non-overlapping spheres.

(c) Generation of two surfaces from the interaction of non-overlapping spheres.

Figure 2.24. Sphere-sphere interactions treated in GEPOL.

A pentakis dodecahedron is then used to mesh each sphere appearing in the molecular system, the molecular atoms and the ghost atoms. By projecting the faces of the pentakis dodecahedron to the surface, a distretization into 60 spherical triangles is achieved for each sphere. These meshes are refined towards the intersections of the spheres where the triangle faces have both, visible and invisible, parts. Only the visible triangles are kept to form the surface mesh.

Several extensions of this algorithm have been developed for the case of geometry optimization. The CSC or SWIG are two examples found in [43, 44, 64]. The CSC guarantees a smooth change of the mesh between two configurations of the molecule, while the SWIG uses switching functions to turn on or off elements which become visible/invisible by a change in the geometry.

2.5.2.2. Isodensity

A different approach, integrated in Gaussian, is to define the cavity by the isosurface of the total electron density, [19]. This method has been named IPCM for isodensity PCM. Since the charge outside the cavity tends rapidly to zero, the surface of the cavity is defined by an isosurface with a small value for the electron density. This has the advantage that the atomic forces are continuous with respect to the position of the atoms and easily integrable. We would like to mention that our mesh generator can also be used to construct parametrizations of such isosurfaces as demonstrated in sec. 5.2.

2.5.2.3. DefPol

A different algorithm was proposed by Pomelli and Tomasi, [54, 55], where the molecule is surrounded by a sphere with a triangular mesh on it. This mesh is then distorted until the points lie on the surface of the molecule. Each point is moved on the connecting line to the centres of the molecule until it lies on the surface. To this end, a characteristic function is used to check when the point has passed inside the molecule. This method

has the advantage that it does not need additional spheres to fill the empty spaces. The centres of the flat triangles are then projected in the normal direction onto the surface. This way, a spherical triangulation of the molecular surface can be generated by solving a system of linear equations.

2.5.2.4. FIXPVA

This method is implemented in GAMESS, [71], and stands for fixed points with variable area. Herein, each sphere is divided into the same number of small triangular elements, with fixed position relative to the centre of the atom they reside on. The difference to the GEPOL and DefPol algorithms is that no additional spheres are introduced and the mesh remains on the single spheres. The area of each element belonging to an atom is a smooth function of the distance to the other atoms. The smooth functions used are the same kind of switching functions used in the CSC and SWIG methods. This way, the geometrical derivatives of the elements can be easily calculated.

2.5.3. Other SES cavity generation algorithms

More attention has been given to the generation of an analytically defined SES in the works [29], [57] and [58]. In both approaches, the intersection circles and arcs are determined and discretized. The same number of boundary points are used on each type of partial surface. The partial surfaces are subsequently independently discretized by triangles. In [29], the surface triangulation is then used to obtain quadrangular patches, suited for the use of the wavelet boundary element method. The disadvantage of the method is that it is not robust and can easily fail. The main advantage of [57, 58] is that it treats self intersecting surfaces, as depicted for example in fig. 2.24c and the interior holes of the molecules correctly, while the algorithm in [29] assumes that these cases do not occur.

2.6. Conclusion

The integral equation is transformed into a system of linear equations whose dimension depends on the number of finite elements used in the discretization of the boundary. The resulting system matrix is, in general, a dense matrix. The boundary element method thus suffers from limitations imposed by the number of matrix elements to be stored and the memory and time requirements of solving the resulting linear system.

Methods have been developed to decrease the computational cost of boundary integral calculations using collocation or Galerkin methods, like the fast multipole method [21], panel clustering [25], adaptive cross approximation [2], or $\mathcal{H}$-matrices [24]. In the next chapter, a different choice of the basis functions will be introduced in order to transform the system matrix into a sparse matrix.

CHAPTER 3

Wavelet Galerkin method

The reason behind the construction and use of a wavelet basis for the Galerkin method to solve boundary integral equations is the fact that the resulting system matrices are quasi-sparse from the start and can be reduced to a sparse form by an *a priori* compression, as described in [15, 30, 31]. Appropriate wavelet bases are biorthogonal spline wavelets since the number of vanishing moments can be chosen independently of the polynomial exactness. They were first introduced on the line in [9], brought onto the interval in [14] and finally onto the surface in [30, 31]. The wavelets on the surface are constructed as tensor products of wavelets on the interval that are transported by a parametrization onto the surface. This is why a parametrization of the surface into a mesh with quadrangular elements needs to be used in order to accommodate the tensor products of wavelets.

3.1. Surface parametrization and inner products

In order to be able to represent the wavelets on the surface, the surface has to be split into smaller quadrangular pieces, also called patches. In this case, the surface elements used for the discretization of the molecular cavity are constructed in such a way that they coincide with the elements obtained by regular subdivision of the patches for the wavelet BEM.

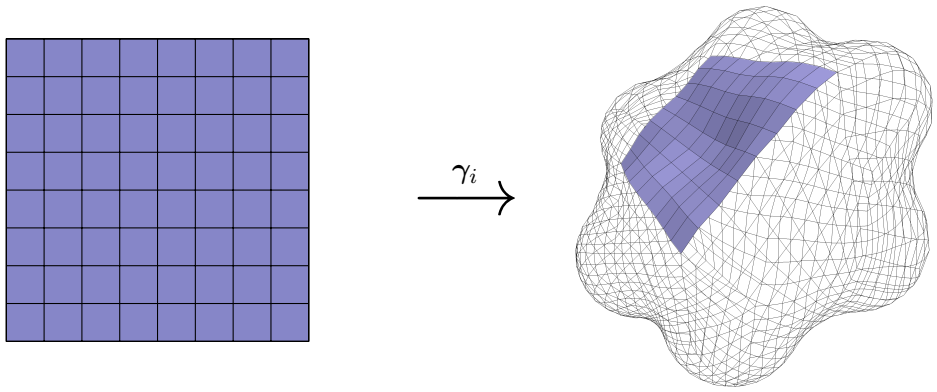


Figure 3.1. Parametric representation of the patch Γ_i by a quadrangular mesh of refinement level 3.

Let the surface Γ be defined by a union of smooth parametric patches. To this end, let $\square := [0,1] \times [0,1]$ denote the unit square. The boundary Γ can then be represented as follows:

$$\Gamma = \bigcup_{i=0}^{n_\square} \Gamma_i \qquad \text{with } \Gamma_i = \boldsymbol{\gamma}_i(\square) \qquad \text{for } i = 0, 1, \ldots n_\square.$$

Herein, $\boldsymbol{\gamma}_i : \square \to \Gamma_i$ is a smooth diffeomorphism from the unit square, $\square$, to the surface patch, Γ_i. The mesh of level n_L is generated by dyadic subdivisions of depth n_L of the unit square into 4^{n_L} elements, which results in a conforming mesh. The intersection of two distinct patches $\Gamma_i \cap \Gamma_j$ is also conforming, which means that the intersection is at most a common vertex or a common edge. A matching condition is hereby imposed. It is required that the parametrization of common edges is the same up to the orientation of the edge. This means that an affine, bijective mapping $\Xi : \square \to \square$ exists such that all points on the common edge between Γ_i and Γ_j satisfy $\boldsymbol{\gamma}_i(\boldsymbol{s}) = \boldsymbol{\gamma}_j(\Xi(\boldsymbol{s}))$. Fig. 3.1 shows such a parametrization for a patch on the surface of a benzene molecule.

For simplicity, let $\Gamma_{i,j,\boldsymbol{k}}$ denote the element $\boldsymbol{k} = (k_1, k_2)$ of patch Γ_i on refinement level j. For each of these elements, one can define the local mapping $\boldsymbol{\gamma}_{i,j,\boldsymbol{k}} : \square \to \Gamma_{i,j,\boldsymbol{k}}$ by the diffeomorphism defined on the patch:

$$\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}(\boldsymbol{s}) = \boldsymbol{\gamma}_i \left(2^{-j} \begin{bmatrix} k_1 + s_1 \\ k_2 + s_2 \end{bmatrix} \right), \qquad \text{where } \boldsymbol{s} = (s_1, s_2) \in \square.$$

This in turn means that the partial derivatives of the local mappings obey the relation

$$\partial_{s_1}^{m_1} \partial_{s_2}^{m_2} \boldsymbol{\gamma}_{i,j,\boldsymbol{k}}(\boldsymbol{s}) = 2^{-j(m_1+m_2)} \partial_{s_1}^{m_1} \partial_{s_2}^{m_2} \boldsymbol{\gamma}_i \left(2^{-j} \begin{bmatrix} k_1 + s_1 \\ k_2 + s_2 \end{bmatrix} \right).$$

The inner product in $L^2(\Gamma)$ can be written in accordance with the patch decomposition:

$$\begin{aligned}\langle u, v\rangle_{L^2(\Gamma)} &= \int_\Gamma u(\boldsymbol{x})v(\boldsymbol{x})\,\mathrm{d}\sigma_{\boldsymbol{x}} = \sum_{i=0}^{n_\square}\int_{\Gamma_i} u(\boldsymbol{x})v(\boldsymbol{x})\,\mathrm{d}\sigma_{\boldsymbol{x}} \\ &= \sum_{i=0}^{n_\square}\int_\square u(\boldsymbol{\gamma}_i(\boldsymbol{s}))v(\boldsymbol{\gamma}_i(\boldsymbol{s}))\left\|\frac{\partial\boldsymbol{\gamma}_i(\boldsymbol{s})}{\partial s_1}\times\frac{\partial\boldsymbol{\gamma}_i(\boldsymbol{s})}{\partial s_2}\right\|\mathrm{d}\boldsymbol{s}, \\ &= \sum_{i=0}^{n_\square}\sum_{\boldsymbol{k}}\int_\square u(\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}(\boldsymbol{s}))v(\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}(\boldsymbol{s}))\kappa_{i,j,\boldsymbol{k}}(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}.\end{aligned}$$

Here, $\kappa_{i,j,\boldsymbol{k}}$ is the surface measure taken at the point $\boldsymbol{s} = (s_1, s_2) \in \square$, which using the diffeomorphism $\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}$ is given by

$$\kappa_{i,j,\boldsymbol{k}}(\boldsymbol{s}) := \left\|\frac{\partial\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}(\boldsymbol{s})}{\partial s_1}\times\frac{\partial\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}(\boldsymbol{s})}{\partial s_2}\right\|.$$

The way of achieving the discretization of the molecular cavity into smooth parametric patches is described in chpt. 4.

3.2. Representation in a wavelet basis

At the core of the wavelet boundary element method lies the definition of a sequence of hierarchical trial spaces, spanned by standard finite element ansatz functions

$$\{0\} := V_{-1} \subset V_0 \subset V_1 \subset \ldots \subset V_{n_L},$$

where $V_j = \operatorname{span}\{\varphi_{j,\boldsymbol{k}} : \boldsymbol{k} \in \Delta_j\}$ and $\varphi_{j,\boldsymbol{k}}$ are the standard finite element ansatz functions defined as introduced in chpt. 2, and Δ_j is an index set. The index set and the ansatz functions on the surface will be described shortly. The spaces V_j are called single-scale spaces. In the wavelet method, the space V_j is split into the direct sum

$$V_j = V_{j-1} \oplus W_j. \tag{3.2}$$

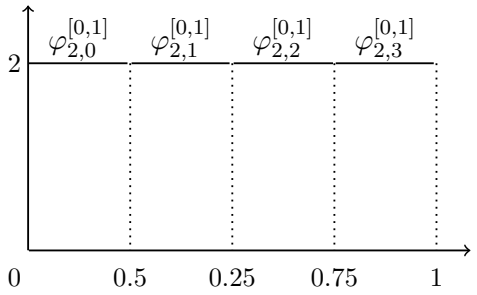
The resulting complementary space W_j is called the wavelet space which is not necessarily orthogonal to V_{j-1}. It is spanned by the wavelet basis:

$$W_j = \operatorname{span}\{\psi_{j,\boldsymbol{k}} : \boldsymbol{k} \in \nabla_j := \Delta_j \setminus \Delta_{j-1}\}.$$

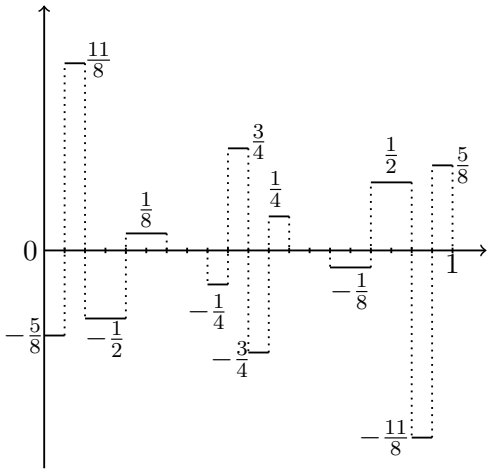
Recursive splitting of the trial spaces leads to the wavelet decomposition $V_{n_L} = \bigoplus_{j=0}^{n_L} W_j$ by inserting eq. (3.2), where $W_0 := V_0$.

3.2.1. Wavelets on the interval

Wavelets on a two-dimensional surface are defined as a tensor product of wavelets on the interval. Therefore, first the wavelets on the interval have to be introduced in order to fully grasp the qualities of the wavelets on the surface. In this thesis, trial spaces of piecewise constant and piecewise bilinear functions will be considered.



(a) Piecewise constant single-scale basis for the interval $[0, 1]$ on refinement level 2.



(b) Piecewise constant wavelet with 3 vanishing moments including the left and right boundary wavelet. Every tick on the x-axis represents the support of one single-scale basis function.

Figure 3.3. Piecewise constant ansatz functions (left) and piecewise constant wavelets with 3 vanishing moments (right).

3.2.1.1. Piecewise constant ansatz functions

For the piecewise constant ansatz functions on the interval $[0, 1]$, the index set is defined as $\Delta_j^{[0,1]} := \{0, 1, \ldots, 2^j - 1\}$. The L^2-normalized piecewise constant ansatz functions then look like

$$\varphi_{j,k}^{[0,1]}(x) = \begin{cases} 2^{j/2}, & x \in [2^{-j}k, 2^{-j}(k+1)], \\ 0, & \text{otherwise.} \end{cases}$$

Fig. 3.3a shows such a choice of basis functions for the refinement level $j = 2$.

The wavelet space, $W_j^{[0,1]}$, defined by the relation $V_j^{[0,1]} = V_{j-1}^{[0,1]} \oplus W_j^{[0,1]}$, is spanned in this case by

$$W_j^{[0,1]} = \operatorname{span}\{\psi_{j,k}^{[0,1]} : k \in \nabla j^{[0,1]} := \{0, 1, \ldots, 2^{j-1} - 1\}\}.$$

The wavelets can be split into interior wavelets and boundary wavelets. The interior wavelets are given by

$$\begin{aligned} \psi_{j,k}^{[0,1]}(x) = &\frac{1}{\sqrt{2}} \left(-\frac{1}{4}\varphi_{j,2k-1}^{[0,1]}(x) + \frac{3}{4}\varphi_{j,2k}^{[0,1]}(x) \right. \\ &\left. -\frac{3}{4}\varphi_{j,2k+1}^{[0,1]}(x) + \frac{1}{4}\varphi_{j,2k+2}^{[0,1]}(x) \right), \qquad k = 1, \ldots, 2^{j-1} - 2. \end{aligned}$$

The left boundary wavelet has the form

$$\psi_{j,0}^{[0,1]}(x) = \frac{1}{\sqrt{2}}\left(-\frac{5}{8}\varphi_{j,0}^{[0,1]}(x) + \frac{11}{8}\varphi_{j,1}^{[0,1]}(x) - \frac{1}{2}\varphi_{j,2}^{[0,1]}(x) - \frac{1}{2}\varphi_{j,3}^{[0,1]}(x) + \frac{1}{8}\varphi_{j,4}^{[0,1]}(x) + \frac{1}{8}\varphi_{j,5}^{[0,1]}(x)\right),$$

while the right boundary wavelet is defined by symmetry:

$$\psi_{j,2^{j-1}-1}^{[0,1]}(x) = -\psi_{j,0}^{[0,1]}(1-x).$$

Fig. 3.3b gives a schematic overview of the wavelets. By construction, these wavelets satisfy the relation

$$\int_0^1 x^r \psi_{j,k}^{[0,1]}(x)\,\mathrm{d}x = 0 \text{ for all } r = 0, 1, 2,$$

and thus the parameter $\tilde{d} = 3$ represents the order of vanishing moments, [27].

3.2.1.2. Piecewise linear ansatz functions

In the case of piecewise linear ansatz functions, the index set is given by the range $\Delta_j^{[0,1]} := \{0, 1, \ldots, 2^j\}$. The L^2-normalized piecewise linear ansatz functions are given by

$$\varphi_{j,k}^{[0,1]}(x) = \begin{cases} 2^{3j/2}(x - 2^{-j}(k-1)), & x \in [2^{-j}(k-1), 2^{-j}k), \\ 2^{3j/2}(2^{-j}(k+1) - x), & x \in [2^{-j}k, 2^{-j}(k+1)), \\ 0, & \text{otherwise}, \end{cases} \qquad \text{and } k = 1, \ldots, 2^j - 1.$$

At the boundary, the ansatz functions take only the part of the interval inside the domain into account

$$\varphi_{j,0}^{[0,1]}(x) = \begin{cases} 2^{3j/2}(2^{-j} - x), & x \in [0, 2^{-j}), \\ 0, & \text{otherwise}. \end{cases}$$

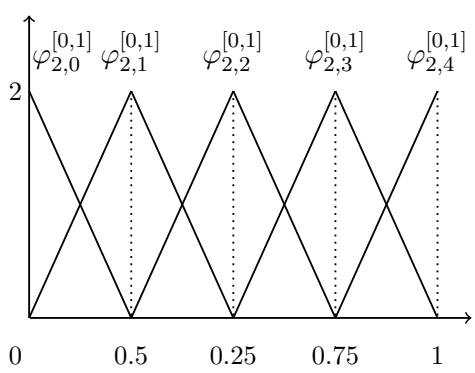
By symmetry, it holds for the basis function at the right boundary

$$\varphi_{j,2^j}^{[0,1]}(x) = \varphi_{j,0}^{[0,1]}(1-x).$$

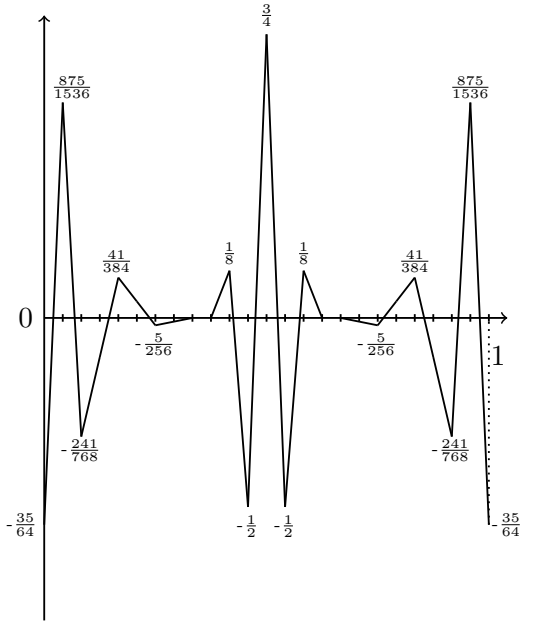
In fig. 3.4a, the discretization using linear single-scale basis functions on a mesh of level 2 is depicted, while fig. 3.4b shows the related wavelet basis.

The wavelet space is similarly defined as for the piecewise constant ansatz functions case by $W_j^{[0,1]} = \operatorname{span}\{\psi_{j,k}^{[0,1]} : k \in \nabla_j^{[0,1]} := \{0, 1, \ldots, 2^j - 1\}\}$. The interior wavelets take the form

$$\psi_{j,k}^{[0,1]}(x) = \frac{1}{\sqrt{2}}\left(\frac{1}{8}\varphi_{j,2k-2}^{[0,1]}(x) - \frac{1}{2}\varphi_{j,2k-1}^{[0,1]}(x) + \frac{3}{4}\varphi_{j,2k}^{[0,1]}(x) - \frac{1}{2}\varphi_{j,2k+1}^{[0,1]}(x) + \frac{1}{8}\varphi_{j,2k+2}^{[0,1]}(x)\right), \qquad k = 1, \ldots, 2^j - 2.$$



(a) Piecewise linear single-scale basis for the interval $[0, 1]$ on refinement level 2.



(b) Piecewise linear wavelets with 4 vanishing moments including the left and right boundary wavelets. Each tick on the x-axis represents one nodal basis function.

Figure 3.4. Piecewise linear ansatz functions (left) and piecewise linear wavelets with 4 vanishing moments (right).

Moreover, the left boundary wavelet looks like

$$\begin{aligned}\psi_{j,0}^{[0,1]}(x) =& \frac{1}{\sqrt{2}}\left(-\frac{35}{64}\varphi_{j,0}^{[0,1]}(x) + \frac{875}{1536}\varphi_{j,1}^{[0,1]}(x) - \frac{241}{768}\varphi_{j,2}^{[0,1]}(x) - \frac{53}{512}\varphi_{j,3}^{[0,1]}(x)\right.\\ &\left.+\frac{41}{384}\varphi_{j,4}^{[0,1]}(x) + \frac{67}{1536}\varphi_{j,5}^{[0,1]}(x) - \frac{5}{256}\varphi_{j,6}^{[0,1]}(x) - \frac{5}{512}\varphi_{j,7}^{[0,1]}(x)\right).\end{aligned}$$

At the right boundary, symmetry can be used again:

$$\psi_{j,2^j-1}^{[0,1]}(x) = \psi_{j,0}^{[0,1]}(1-x).$$

By construction, these wavelets have $\tilde{d} = 4$ vanishing moments, meaning that

$$\int_0^1 x^r \psi_{j,k}^{[0,1]}(x)\,\mathrm{d}x = 0 \text{ for all } r = 0,1,2,3.$$

3.2.2. Wavelets on the surface

The ansatz spaces on the unit square can be defined by using the tensor product of the univariate spaces, i.e. $V_j^{\square} = V_j^{[0,1]} \otimes V_j^{[0,1]}$. The ansatz space is thus spanned by tensor products of the basis on the interval, $V_j^{\square} = \operatorname{span}\{\varphi_{j,\boldsymbol{k}}^{\square} : \boldsymbol{k} \in \Delta_j^{\square}\}$, with the index set given by $\Delta_j^{\square} := \Delta_j^{[0,1]} \times \Delta_j^{[0,1]}$ and the scaling functions $\varphi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{x}) := \varphi_{j,k_1}^{[0,1]}(x_1)\varphi_{j,k_2}^{[0,1]}(x_2)$, where $\boldsymbol{x} = (x_1, x_2) \in \square := [0,1] \times [0,1]$.

By inserting the direct sum (3.2) in the representation, one gets

$$\begin{aligned}V_j^{\square} &= V_j^{[0,1]} \otimes V_j^{[0,1]} = V_j^{[0,1]} \otimes \left(V_{j-1}^{[0,1]} \oplus W_j^{[0,1]}\right)\\ &= \left(V_j^{[0,1]} \otimes V_{j-1}^{[0,1]}\right) \oplus \left(V_j^{[0,1]} \otimes W_j^{[0,1]}\right)\\ &= \left(\left(V_{j-1}^{[0,1]} \oplus W_j^{[0,1]}\right) \otimes V_{j-1}^{[0,1]}\right) \oplus \left(V_j^{[0,1]} \otimes W_j^{[0,1]}\right)\\ &= V_{j-1}^{\square} \oplus \left(W_j^{[0,1]} \otimes V_{j-1}^{[0,1]}\right) \oplus \left(V_j^{[0,1]} \otimes W_j^{[0,1]}\right).\end{aligned}$$

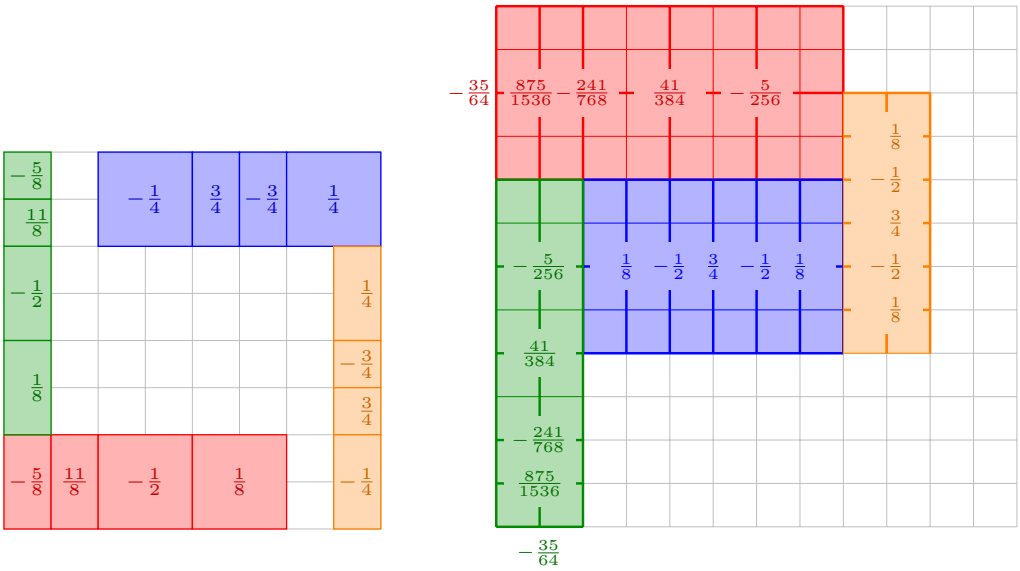
Hence, the definition of the wavelet space on the surface is found,

$$W_j^{\square} = \left(W_j^{[0,1]} \otimes V_{j-1}^{[0,1]}\right) \oplus \left(V_j^{[0,1]} \otimes W_j^{[0,1]}\right),$$

and one arrives at $V_j^{\square} = V_{j-1}^{\square} \oplus W_j^{\square}$. This already defines which elements need to be added to end up with a direct sum for the refinement from the level $V_{j-1}^{\square}$ to $V_j^{\square}$. It leads to wavelets stemming from the combinations

$$\begin{aligned}W_j^{[0,1]} \otimes V_{j-1}^{[0,1]} &\Rightarrow \psi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{x}) = \psi_{j,k_1}^{[0,1]}(x_1)\ \varphi_{j-1,k_2}^{[0,1]}(x_2),\\ V_j^{[0,1]} \otimes W_j^{[0,1]} &\Rightarrow \psi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{x}) = \varphi_{j,k_1}^{[0,1]}(x_1)\ \psi_{j,k_2}^{[0,1]}(x_2),\end{aligned}$$

for $\boldsymbol{x} = (x_1, x_2) \in \square$ and appropriate indices $\boldsymbol{k} = (k_1, k_2)$. In particular, for the first type of wavelets, k_1 is a wavelet index from $\nabla_j^{[0,1]}$ and k_2 a scaling function index



(a) Piecewise constant wavelets with 3 vanishing moments on the square. Type 1 wavelets: boundary wavelet and interior wavelet and type 2 wavelets: boundary wavelet and interior wavelet are depicted.

(b) Piecewise bilinear wavelets with 4 vanishing moments on the square. Type 1 wavelets: boundary wavelet and interior wavelet and type 2 wavelets: boundary wavelet and interior wavelet are depicted.

Figure 3.5. Schematic representation of piecewise constant wavelets (left) and piecewise bilinear wavelets (right) on the unit square. The numbers represent the coefficients of the respective single-scale basis functions, while the color represents the wavelet's support. For the piecewise bilinear case, the coefficients would have to be linearly interpolated on the support.

from $\Delta_{j-1}^{[0,1]}$ while, for the second type of wavelets, $k_1 \in \Delta_j^{[0,1]}$ and $k_2 \in \nabla_j^{[0,1]}$. The resulting two-dimensional wavelets are shown schematically in fig. 3.5a and fig. 3.5b for the piecewise constant and the piecewise bilinear case, respectively.

By lifting the square onto the patches through the mappings $\boldsymbol{\gamma}_i$, one finally gets the representation of the basis on the surface. The evaluation of the single-scale basis or the wavelets in a surface point, $\boldsymbol{x} \in \Gamma_i$, is done by making use of the inverse diffeomorphism to transform the point from the surface to the unit square, $\boldsymbol{\gamma}_i^{-1}(\boldsymbol{x}) \in \square$. The single-scale basis function and the wavelet defined on the unit square are then evaluated. The evaluation looks like

$$\begin{aligned} \varphi_{i,j,\boldsymbol{k}}(\boldsymbol{x}) &= \varphi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{\gamma}_i^{-1}(\boldsymbol{x})), \qquad \boldsymbol{k} \in \Delta_j^{\square}, \\ \psi_{i,j,\boldsymbol{k}}(\boldsymbol{x}) &= \psi_{j,\boldsymbol{k}}^{\square}(\boldsymbol{\gamma}_i^{-1}(\boldsymbol{x})), \qquad \boldsymbol{k} \in \nabla_j^{\square}, \end{aligned}$$

where $i = 0, \ldots, n_{\square}$ are the patches used to represent the surface and $\boldsymbol{x} \in \Gamma_i$.

3.3. Compression of the system matrix

The main advantage of choosing the wavelet basis and not the standard single-scale basis is described in [15], where a compression rule is presented that defines the sparsity pattern of the system matrix without computing any integrals. It only employs the properties of the wavelets and the elements in their support. This *a priori* compression is done in two steps. First, contributions stemming from wavelets that are far enough from each other are discarded, leaving only $O(N_{n_L} \log(N_{n_L}))$ non-zero entries, with N_{n_L} being the number of degrees of freedom for a uniform refinement on level n_L. This is carried out in a recursive fashion starting at the coarse level. Secondly, even more elements of the system matrix by eliminating entries corresponding to wavelets that lie in the smooth part of another wavelet. The number of non-zero matrix entries is reduced to $O(N_{n_L})$. After applying these two *a priori* compression steps, an *a posteriori* compression can be used to further reduce entries which have a very small contribution.

In order to understand the compression rules, we denote the *convex hull* of the support of the wavelet $\psi_{i,j,\boldsymbol{k}}$ by

$$\Omega_{i,j,\boldsymbol{k}} = \text{conv hull } (\operatorname{supp} \psi_{i,j,\boldsymbol{k}}) \tag{3.6}$$

and the *singular support* of the wavelet $\psi_{i,j,k}$ by

$$\Omega'_{i,j,\boldsymbol{k}} = \operatorname{sing\,supp} \psi_{i,j,\boldsymbol{k}}. \tag{3.7}$$

The singular support contains all the points in the support where the wavelet is non-smooth. In the implementation, the convex hull of the support of the wavelets is taken as the sphere that encloses all elements in the support of the wavelet at hand. These bounding boxes can also easily be used to calculate the distance between the singular support of different wavelets if the wavelets are far away from each other. Otherwise, the distance can be computed element by element on the unit square if the support of the wavelets intersect.

Since the wavelets are expressed as linear combinations of single-scale basis functions, there is a direct link between the wavelets and the elements used for the spatial discretization. An index is stored in the wavelet structure towards all elements in the linear combination and, likewise, an index is stored in the element structure towards all wavelets which have the element in their support. The refinement of the quadrangular mesh naturally induces an inheritance relation between elements. One element on level j is subdivided into four elements on level $j+1$. This will be reflected in the wavelet structure by introducing a parent-child relation between wavelets of consecutive levels. A wavelet $\psi_{i,j+1,\boldsymbol{k}}$ on level $j+1$ is said to be the child of a wavelet $\psi_{i,j,\boldsymbol{k}'}$ on level j, if the elements in the support of the wavelet $\psi_{i,j+1,\boldsymbol{k}}$ are included in the refinement of the elements of the parent wavelet, $\psi_{i,j,\boldsymbol{k}'}$. The compression rule makes use of these relations in choosing which matrix entries are relevant. For example, if two wavelets on the coarse level do not interact due to the distance between their support, their children will also not produce a relevant matrix entry. That way, linear complexity can be realized, see [15, 31] for details.

3.3.1. A priori compression

The *a priori* compression pre-calculates which entries of the system matrix have a relevant contribution to the solution. It is governed by the following set of compression rules

$$(\boldsymbol{A}_{n_L})_{(i,j,\boldsymbol{k}),(i',j',\boldsymbol{k}')} = \begin{cases} 0, & \operatorname{dist}(\Omega_{i,j,\boldsymbol{k}}, \Omega_{i',j',\boldsymbol{k}'}) > B_{j,j'} \text{ and } j, j' > 0, \\ 0, & \operatorname{dist}(\Omega'_{i,j,\boldsymbol{k}}, \Omega_{i',j',\boldsymbol{k}'}) > B'_{j,j'} \text{ if } j' > j \geq 0, \\ & \operatorname{dist}(\Omega_{i,j,\boldsymbol{k}}, \Omega'_{i',j',\boldsymbol{k}'}) > B'_{j,j'} \text{ if } j > j' \geq 0, \\ \langle \boldsymbol{A}\psi_{i',j',\boldsymbol{k}'}, \psi_{i,j,\boldsymbol{k}} \rangle_{L^2(\Gamma)}, & \text{otherwise,} \end{cases} \tag{3.8}$$

where $\operatorname{dist}(\cdot,\cdot)$ denotes the distance, either between the convex hull of the wavelets or between the singular support and the convex hull, respectively. The first and second conditions represent the first and second compression step, respectively. The parameters j, j' are the levels of the wavelets under consideration and the level-dependent cut-off parameters $B_{j,j'}$ and $B'_{j,j'}$ are given by

$$B_{j,j'} = a \max \left\{ 2^{-\min\{j,j'\}}, 2^{\frac{n_L(2d'-2q)-(j+j')(d'+\tilde{d})}{(2\tilde{d}+2q)}} \right\},$$
$$B'_{j,j'} = a \max \left\{ 2^{-\min\{j,j'\}}, 2^{\frac{n_L(2d'-2q)-(j+j')d'-\max\{j,j'\}\tilde{d}}{(\tilde{d}+2q)}} \right\},$$

where $2q$ is the order of the integral operator under consideration, as found in lemma 2.15. The compression parameters a and d' can be chosen in the ranges

$$a \geq 1, \qquad d < d' < \tilde{d} + 2q. \tag{3.9}$$

Therein, d is the approximation order of the trial spaces, i.e. $d = 1$ for piecewise constant wavelets and $d = 2$ for piecewise bilinear wavelets, respectively. The integer

$\tilde{d}$ stands for the number of vanishing moments of the wavelet basis, which is $\tilde{d} = 3$ for the piecewise constant wavelets and $\tilde{d} = 4$ for the piecewise bilinear wavelets.

The first compression rule neglects the matrix entry resulting from two wavelets, $\psi_{i,j,\boldsymbol{k}}$ and $\psi_{i',j',\boldsymbol{k}'}$, on level j and j', if the distance between the supports, $\operatorname{dist}(\Omega_{i,j,\boldsymbol{k}}, \Omega_{i',j',\boldsymbol{k}'})$, is large enough. By construction, all children have their support included in the ones of the parents. This means that all the entries that result from the children of $\psi_{i,j,\boldsymbol{k}}$ and $\psi_{i',j',\boldsymbol{k}'}$ can also be ignored, since $B_{j,j'} \geq B_{\tilde{j},\tilde{j}'}$ for all $\tilde{j} \geq j$ and $\tilde{j}' \geq j'$. For the second compression, the refinement structure of the wavelet tree can again be used. Let $\psi_{i,j,\boldsymbol{k}}$ be in the smooth part of $\psi_{i',j',\boldsymbol{k}'}$. Wavelets that have their support included in $\Omega_{i,j,\boldsymbol{k}}$ will also lie in the smooth part of $\psi_{i',j',\boldsymbol{k}'}$. This means that the interaction between the children of $\psi_{i,j,\boldsymbol{k}}$ and $\psi_{i',j',\boldsymbol{k}'}$ can be neglected, since $B'_{j,j'} \geq B'_{\tilde{j},j'}$ for all $\tilde{j} \geq j \geq j'$ and likewise $B_{j,j'} \geq B_{j,\tilde{j}'}$ for all $\tilde{j}' \geq j' \geq j$, respectively. Thus, the compression pattern of the system matrix can be calculated hierarchically in optimal complexity by starting from the coarsest level.

3.3.2. A posteriori compression

The *a priori* compression already sets the sparsity pattern for the system matrix. After computing the entries of the system matrix, the *a posteriori* compression is applied. It leaves out other sufficiently small entries by the rule

$$(\boldsymbol{A}_{n_L})_{(i,j,\boldsymbol{k}),(i',j',\boldsymbol{k}')} \begin{cases} 0, & \text{if } \left|(\boldsymbol{A}_{n_L})_{(i,j,\boldsymbol{k}),(i',j',\boldsymbol{k}')}\right| \leq \epsilon_{j,j'}, \\ (\boldsymbol{A}_{n_L})_{(i,j,\boldsymbol{k}),(i',j',\boldsymbol{k}')}, & \text{otherwise,} \end{cases} \tag{3.10}$$

where the level dependent coefficients $\epsilon_{j,j'}$ are given by

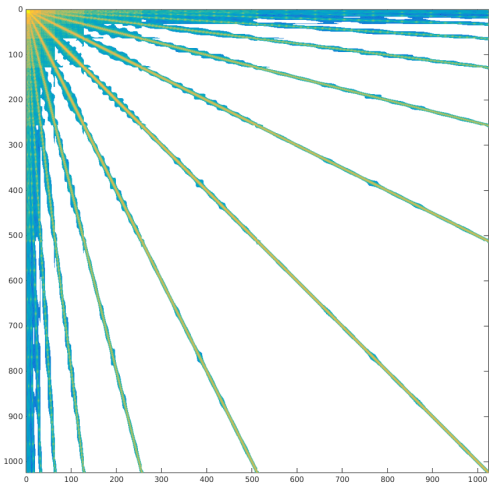
$$\epsilon_{j,j'} = b \min \left\{ 2^{-|j-j'|}, 2^{-(2n_L-(j+j'))\frac{d'-q}{\tilde{d}+q}} \right\} 2^{-(d'-q)(2n_L-(j+j'))}. \tag{3.11}$$

The *a posteriori* compression parameters have to satisfy $b < 1$, while $d < d' < \tilde{d} + 2q$ like for the *a priori* compression.

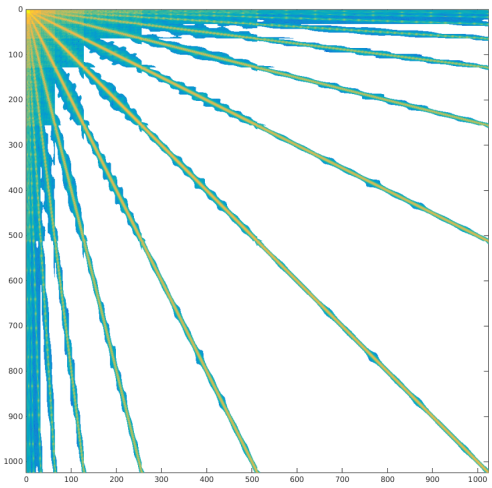
3.3.3. Influence of the compression

To understand the influence of the *a priori* and *a posteriori* compression, we will discuss the matrix arising from the single layer operator on the boundary of a two-dimensional domain, discretized on level $n_L = 10$.

The behavior of the *a priori* compression parameters are illustrated in fig. 3.12 and fig. 3.13, which show the patterns obtained by applying only the first or the full *a priori* compression, respectively. One can see that the parameter a has an influence on the overall accuracy of the matrix. This is reflected in the thickness of the sparsity pattern rays, as depicted in fig. 3.12. The parameter d' is directly related to the precision on the coarse level. This, however, cannot easily be observed with the naked eye. Tab. 3.14 summarizes the number of non-zero elements of the system matrix

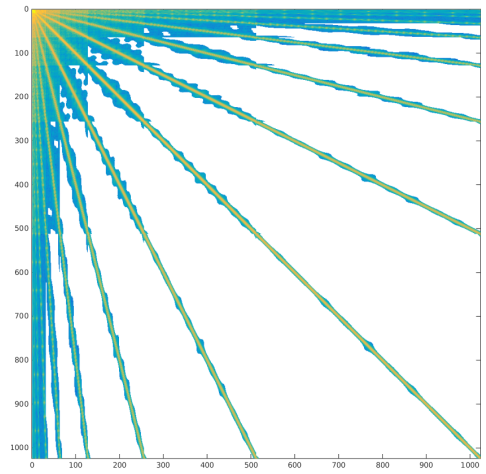


(a) $a = 1.0,\ d' = d$
nnz = 158 540 (15%)

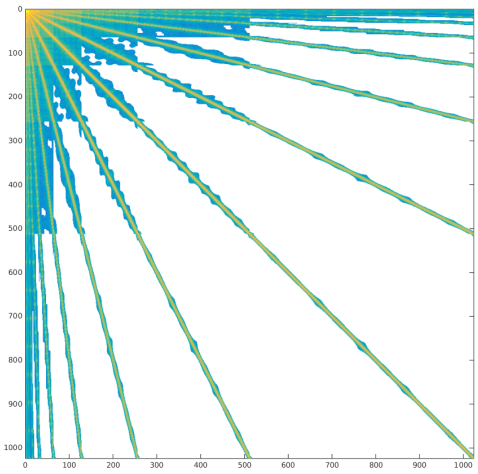


(b) $a = 2.0,\ d' = d$
nnz = 205 878 (19%)

Figure 3.12. Influence on the sparsity pattern for the system matrix representing the single layer operator $\mathcal{V}$ in a piecewise linear wavelet basis of the first *a priori* parameter a. The parameter d' is kept fixed at its minimum value: $d' = d$. The number of non-zero elements (nnz) is reported below each matrix.



(a) First *a priori* compression.
nnz = 221 716 (21%)



(b) Full *a priori* compression.
nnz = 204 404 (19%)

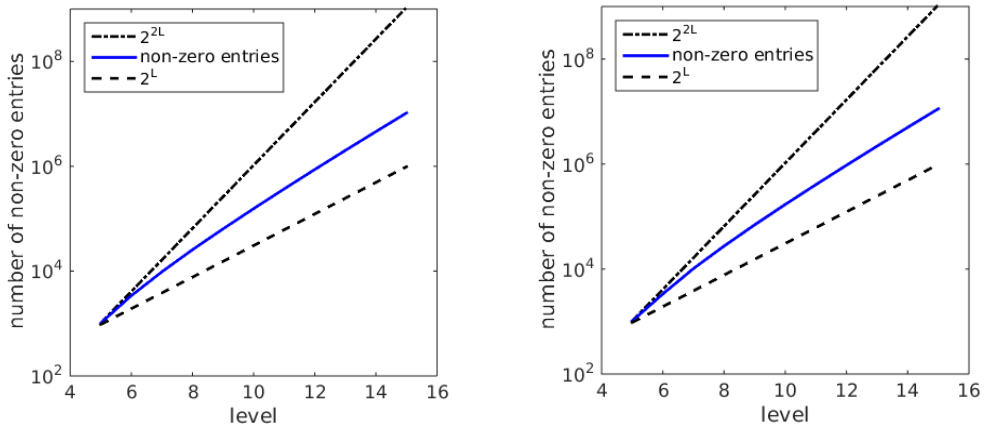
Figure 3.13. Influence on the sparsity pattern for the system matrix representing the single layer operator $\mathcal{V}$ in a piecewise linear wavelet basis of the first and second *a priori* compression. Both parameters are kept fixed: $a = 2.0$ and $d' = d'_{1/2}$. The number of non-zero elements (nnz) is reported below each matrix.

depending on the *a priori* compression parameters. The impact of the first and second *a priori* compressions is also summarized. In all cases, the relevant part is the first *a priori* compression which discards 80% of the entries on average, setting the sparsity pattern of the system matrix. The second compression discards about 8% additional negligible entries.

		d	$d'_{1/2}$	$d'_{\max}$
First *a priori* compression	$a = 1$	158 540	169 550	180 022
	$a = 2$	205 878	221 716	236 350
Full *a priori* compression	$a = 1$	136 456	155 214	170 082
	$a = 2$	181 470	204 404	227 930

Table 3.14. Influence of the *a priori* compression parameters on the number of non-zero elements for the single layer operator $\mathcal{V}$ in a piecewise linear wavelet basis. The value $d'_{1/2} = \frac{\tilde{d}+2q+d}{2}$ is selected as an intermediate value for d'. The total number of elements of the full matrix would be 1 048 576.

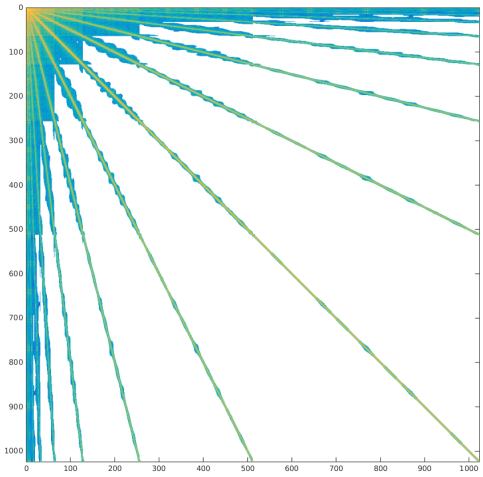
The memory requirement for both wavelet bases chosen grows linearly with the refinement level as shown in [15] and fig. 3.15. Fig. 3.16 shows a comparison of the sparsity pattern for piecewise constant and piecewise linear wavelets.



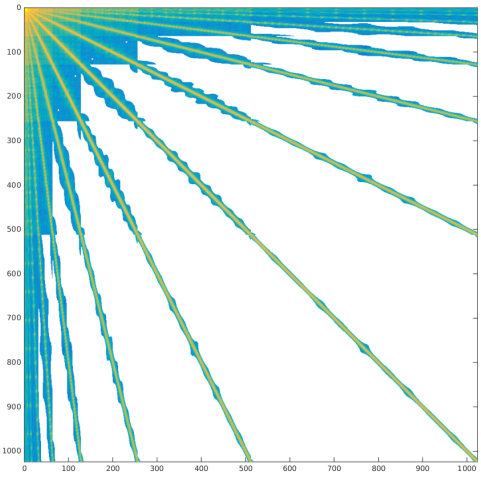
(a) Piecewise constant wavelets. **(b)** Piecewise linear wavelets.

Figure 3.15. Number of non-zero entries in the matrix with increasing level for piecewise constant wavelets (left) and piecewise linear wavelets (right).

The same two-dimensional example is being used to illustrate the influence of the *a posteriori* compression parameters. Tab. 3.17 summarizes the number of non-zero elements for different choices of the *a posteriori* parameters b and d'. The influence of the parameters for the *a posteriori* compression is not as large as the influence of the *a priori* compression. The *a posteriori* compression discards an average of 3% additional entries in the system matrix. This is also illustrated in fig. 3.18 where the effect of the *a posteriori* compression is shown.



(a) Piecewise constant wavelets.
nnz = 155 450 (15%)



(b) Piecewise linear wavelets.
nnz = 236 350 (23%)

Figure 3.16. Comparison between the sparsity pattern of the first *a priori* compression for the piecewise constant (left) and for the piecewise linear (right) wavelet bases with parameters $a = 2.0$ and $d' = d'_{\max}$.

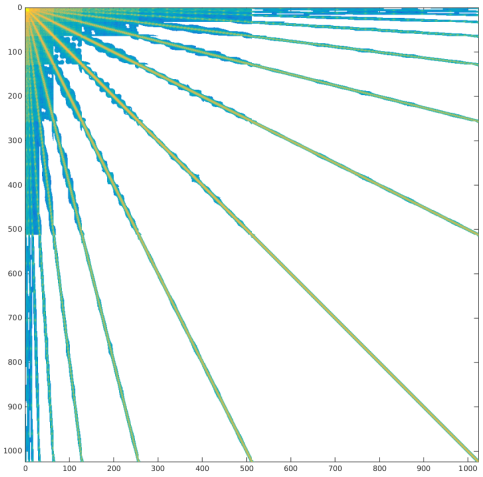
	d	$d'_{1/2}$	$d'_{\max}$
$b = 0.1$	121 678	145 782	163 602
$b = 0.01$	131 852	152 218	168 186
$b = 0.001$	135 642	154 634	169 920
$b = 0$	136 456	155 214	170 082

Table 3.17. Influence of the *a posteriori* compression parameters on the number of non-zero elements for the single layer operator $\mathcal{V}$ in a piecewise linear wavelet basis. The value $d'_{1/2} = \frac{\tilde{d}+2q+d}{2}$ is selected as an intermediate value for d'. The parameter for the first compression is kept fixed $a = 1$. The total number of elements of the full matrix would be 1 048 576.

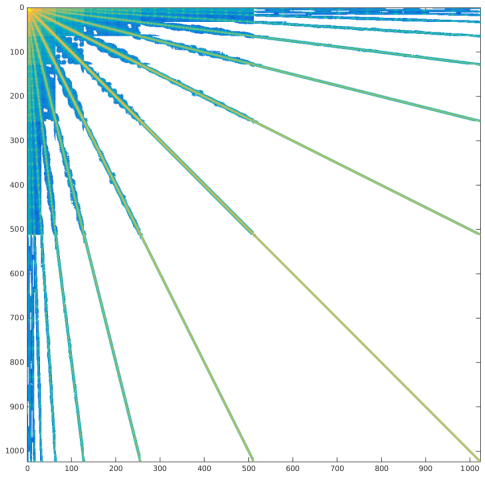
3.4. Implementation details

3.4.1. Control flow

The wavelet solver is a versatile tool that can be applied to any problem where the fundamental solution and the parametrization of the surface are known. The control flow diagram of the solver can be seen in fig. 3.19. The parameter that influences the precision of the wavelet solver that is fixed from the beginning is the level of refinement of the surface, n_L, provided by the input file from the cavity generator. The approximation order, d, and the number of vanishing moments, $\tilde{d}$, are set by the ansatz functions chosen. The parameters that can be adjusted are the compression parameters a, b, d', and the precision of the iterative solver, ϵ. For all the details of the implementation of the wavelet boundary element code, see [31].



(a) Full *a priori* compression parameters $d' = d'_{1/2}$, $b = 0.001$
nnz = 154 634 (15%)



(b) Full *a priori* and *a posteriori* compression parameters $d' = d'_{1/2}$, $b = 0.1$
nnz = 145 782 (14%)

Figure 3.18. Influence on the sparsity pattern for the system matrix representing the single layer operator $\mathcal{V}$ in a piecewise linear wavelet basis of the full *a priori* compression and the *a posteriori* compression. The parameter for the *a priori* compression is set to $a = 1$. The number of non-zero elements (nnz) is reported below each matrix.

3.4.1.1. Initialization

The wavelet solver starts off by reading in the vertices of the elements in a patchwise manner and the parameters for the compression. The first step is to calculate an appropriate interpolation of the surface points on each patch. This interpolation is used as the diffeomorphism, γ_i, from the unit square to the surface. It can then be used to calculate the quadrature points and the tangential and normal derivatives needed in the calculation of the system matrix entries.

The number of the polynomials used is determined by the uniform refinement level, n_L, and the degree, *degree*, required for the polynomials. The formula $2^{n_L} \mod degree = 0$ has to hold. This way, the interpolation divides each patch into disjoint polynomials covering the entire patch. A tensorized Newton interpolation is used for each polynomial. In one dimension, the Newton interpolation through the points $(s_0, \boldsymbol{p}_0), (s_1, \boldsymbol{p}_1), \ldots, (s_n, \boldsymbol{p}_n)$ is given by

$$\boldsymbol{P}^s(x) = [\boldsymbol{p}_0] + [\boldsymbol{p}_0, \boldsymbol{p}_1](x - s_0) + \ldots + [\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_n](x - s_0)(x - s_1)\ldots(x - s_n)$$

with $[\boldsymbol{p}_0, \ldots, \boldsymbol{p}_i]$ being the divided differences obeying the rules

$$\begin{aligned} [\boldsymbol{p}_i] &:= \boldsymbol{p}_i, \\ [\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+l}] &:= \frac{[\boldsymbol{p}_{i+1}, \ldots, \boldsymbol{p}_{i+l}] - [\boldsymbol{p}_i, \ldots, \boldsymbol{p}_{i+l-1}]}{s_{i+l} - s_i}. \end{aligned}$$

This can be generalized to a two-dimensional setting. For a given surface patch, Γ_i,

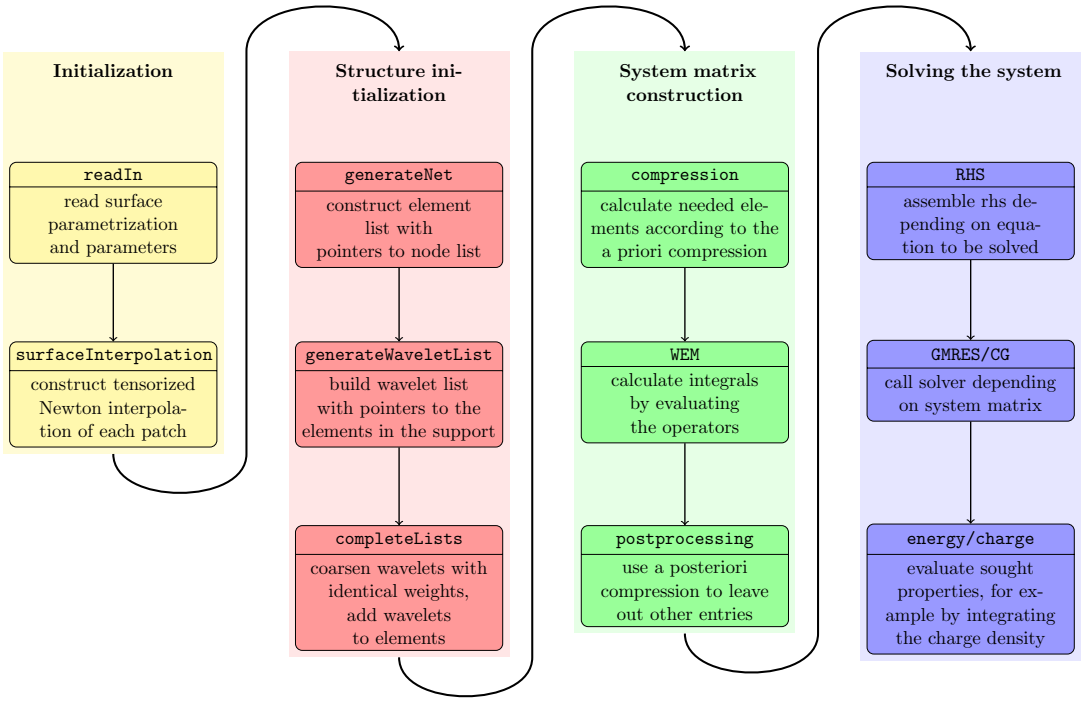


Figure 3.19. Control flow for the wavelet solver.

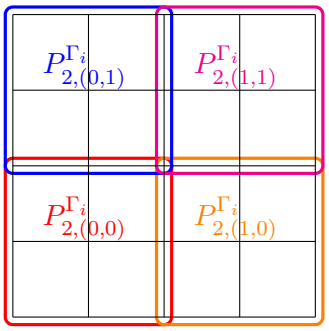


Figure 3.20. Example of two-dimensional interpolation. The number of refinements of the underlying patch is $n_L = 2$ and the degree of the polynomials in each coordinate is $degree = 2$.

the interpolation polynomial through the gridpoints

$$\left((s_0,t_0),\boldsymbol{p}_{(0,0)}\right),\ldots,\left((s_{k_1},t_{k_2}),\boldsymbol{p}_{(k_1,k_2)}\right),\ldots,\left((s_n,t_n),\boldsymbol{p}_{(n,n)}\right)$$

is obtained by the formula

$$\begin{aligned}\boldsymbol{P}^{\Gamma_i}_{n,\tilde{\boldsymbol{k}}}(x_1,x_2) = &[\boldsymbol{P}^s_{n,t_0}(x_1)] + [\boldsymbol{P}^s_{n,t_0}(x_1),\boldsymbol{P}^s_{n,t_1}(x_1)](x_2-t_0)+\ldots\\ &+[\boldsymbol{P}^s_{n,t_0}(x_1),\boldsymbol{P}^s_{n,t_1}(x_1),\ldots,\boldsymbol{P}^s_{n,t_n}(x_1)](x_2-t_0)(x_2-t_1)\ldots(x_2-t_n)\end{aligned}$$

with $\boldsymbol{P}^s_{n,t_i}(x_1)$ being the one-dimensional interpolation polynomial in s-direction with constant $t=t_i$ and the index $\tilde{\boldsymbol{k}}$ a two-dimensional index that identifies the polynomial uniquely. In fig. 3.21, an illustration of the construction of one polynomial can be seen. First, the interpolation is done in the s-direction and then the resulting polynomials are interpolated in the t-direction.

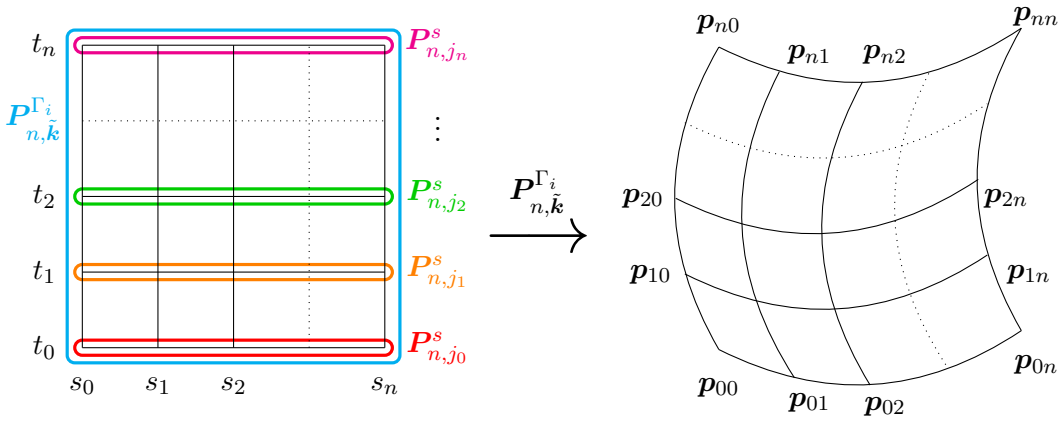


Figure 3.21. Construction of the two-dimensional interpolation of $degree = 4$. Note that the following equalities hold: $t_0 = j_0 \cdot h = j \cdot \tilde{h}$, $t_i = i \cdot h$ and $t_n = (j+1)\cdot\tilde{h}$. Here, h is the mesh size, meaning $2^{n_L}\cdot h = 1$, and $\tilde{h}$ is the step size for the polynomials, meaning $2^{n_L}\cdot\tilde{h} = degree$.

In fig. 3.20, the interpolation of degree 2 on a grid of refinement level 2 is illustrated for a single patch. The evaluation of the local mapping $\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}$ with $\boldsymbol{k} = (k_1,k_2)$ is thus approximated by the evaluation of the polynomial $\boldsymbol{P}^{\Gamma_i}_{degree,\tilde{\boldsymbol{k}}}$, where the index $\tilde{\boldsymbol{k}}$ is given by $\tilde{\boldsymbol{k}} = (k_1/degree, k_2/degree)$. The Horner scheme, described in [70], can be used to evaluate the polynomials and their derivatives in a stable way by factoring out common differences. For the one-dimensional example, it would look like:

$$\begin{aligned}\boldsymbol{P}(x) = &\Big(\cdots\big([\boldsymbol{p}_0,\ldots,\boldsymbol{p}_n](x-s_{n-1})+[\boldsymbol{p}_0,\ldots,\boldsymbol{p}_{n-1}]\big)(x-s_{n-2})+\\ &+\ldots+[\boldsymbol{p}_0,\boldsymbol{p}_1]\Big)(x-s_0)+[\boldsymbol{p}_0].\end{aligned}$$

For the two-dimensional case, this is done for each polynomial in s-direction and for the interpolation of the polynomials in t-direction analogously.

3.4.1.2. Structure initialization

The structure initialization block from fig. 3.19 starts with the definition and initialization of an element tree. The element tree replicates the refinement strategy of the mesh. For one element, e, on level j, we store a reference to the four elements resulting from the refinement from level j to level $j+1$. In a forest fashion, each root node is assigned the information of one patch. Each entry in the tree contains the indices of the vertices of the element of the patches and four pointers to the four children stemming from one refinement. Since this code works with a uniform refinement, the resulting element trees are well-balanced.

Similarly, a wavelet tree can be constructed with regard to the elements in the support of the wavelets. The children of a wavelet are the wavelets whose supports are included in the refinement of the elements in the support of the parent wavelet. This means that $\psi_{i,j+1,\boldsymbol{k}}$ is a child of $\psi_{i,j,\boldsymbol{k}}$ if $\Omega_{i,j+1,\boldsymbol{k}} \subset \Omega_{i,j,\boldsymbol{k}}$. Using this information, the wavelet tree is initialized by starting with the coarsest level, i.e. the scaling functions on the patches. The entries in the wavelet tree contain also the weights employed in the representation of the wavelet as a linear combination of single-scale basis functions, as represented schematically in fig. 3.5.

The last block is concerned with associating to one element all the wavelet indices which contain the element in their support. Additionally, wavelets that do not contribute new weights on a lower level element are replaced with wavelet weights for the element on the coarser level. This is done by checking the value of the weights of the children of each element, e, in the support of the wavelet. If the values of the weights corresponding to the single-scale functions of the children of e are equal, the contribution is coarsened to the contribution of e.

3.4.1.3. System matrix construction

Having the element and wavelet trees at hand, the *a priori* compression found in eq. (3.8) can be applied to mark which entries of the system matrix need to be computed. Two matrices are assembled synchronously, the matrix corresponding to the single layer operator, $\mathcal{V}$, and the matrix corresponding to the double layer operator, $\mathcal{K}$, for all the needed boundary integral operators as defined in lemma 2.15. The system matrix is never assembled in a dense way. The entries that need to be computed are calculated by applying the appropriate boundary integral operator in an element-wise fashion and by applying a tensor product Gaussian quadrature on the unit square. Singular and nearly singular integrals are treated as in [63]. The *a posteriori* compression found in eq. (3.10) can then be applied to further reduce the number of elements of the system matrix.

3.4.1.4. Solving the system

Assembling the right hand side of the equation is done by applying the corresponding boundary integral operators in the order that they appear in the integral equations sum-

marized in lemma 2.15. When more than one operator needs to be applied, the Gramian matrix, $\boldsymbol{G}_{n_L} = \big(\langle\varphi_i, \varphi_j\rangle_{L^2(\Gamma)}\big)_{i,j}$, is used in-between to project into the single-scale basis. The Newton potential, $\mathcal{N}_\rho$, is assumed to be known. The Restarted Generalized Minimal Residual (GMRES) iterative solver described in [62] is then used, restarted after 100 iterations, in order to compute the right hand side.

Since the system matrix is ill conditioned if the operator has an order different from zero, $\operatorname{cond} \boldsymbol{A}_{n_L} \sim 2^{2n_L|q|}$, a preconditioner as introduced in [26] is applied. This preconditioner is based on the norm equivalences of the wavelet basis:

$$\|u\|^2_{H^q(\Gamma)} \sim \sum_i \sum_j \sum_{\boldsymbol{k}} 2^{2qj} |\langle u, \psi_{i,j,\boldsymbol{k}}\rangle_{L^2(\Gamma)}|^2, \qquad q \in (-1/2, 1/2).$$

It combines the diagonal matrix $(\boldsymbol{D}^q_{n_L})_{(j,k),(j',k')} := 2^{qj}\delta_{j,j'}\delta_{k,k'}$ with the Gramian matrix, $\boldsymbol{G}_{n_L}$. The Gramian matrix is represented in the single-scale basis and needs to be transformed into the wavelet bases by means of the fast wavelet transform, see [26] for more details.

The linear system is then solved, either by applying a conjugate gradient (CG) solver, [33], in the case of a symmetric and positive definite system matrix like in the standard PCM case from eq. (1.31), or by applying again the GMRES solver for the equations (1.24), (1.28), and (1.32).

3.4.2. Class structure

Fig. 3.22 shows the class diagram for the wavelet classes. It describes a C++ abstract implementation of a generic class `GenericAnsatzFunction`, which contains all aspects of the code that do not depend on the particular choice of ansatz functions. For example, the function generating the element tree does not depend on the choice of basis functions and is located in the generic class. Both compressions, *a priori* and *a posteriori* compression, only depend on the parameters a, b, $\tilde{d}$, and d'. The method is implemented in the `GenericAnsatzFunction` and the parameters are initialized as class members in the respective derived classes.

Two derived classes have been implemented, the `ConstantAnsatzFunction` for the piecewise constant ansatz functions and the `LinearAnsatzFunction` for the piecewise bilinear ansatz functions. The derived classes initialize the constants which are specific to their definition and implement the quadrature routines and the wavelet tree generation. Among the constants and parameters are those relevant for the compression. The quadrature is carried out on the reference element and employs the bijective mapping, $\boldsymbol{\gamma}_{i,j,\boldsymbol{k}}$, from the unit square to the surface element under consideration. It distinguishes between four cases, namely the integration on the same element or the integration of two elements having a common edge, a common vertex, or nothing in common at all. For the last three cases, the integral contains the singularity and the Duffy trick can be used to calculate the value of the integral, see [26] and [63] for more details. The method `simplifyWaveletList` is the one concerned with the coarsening of the wavelets with identical weights.

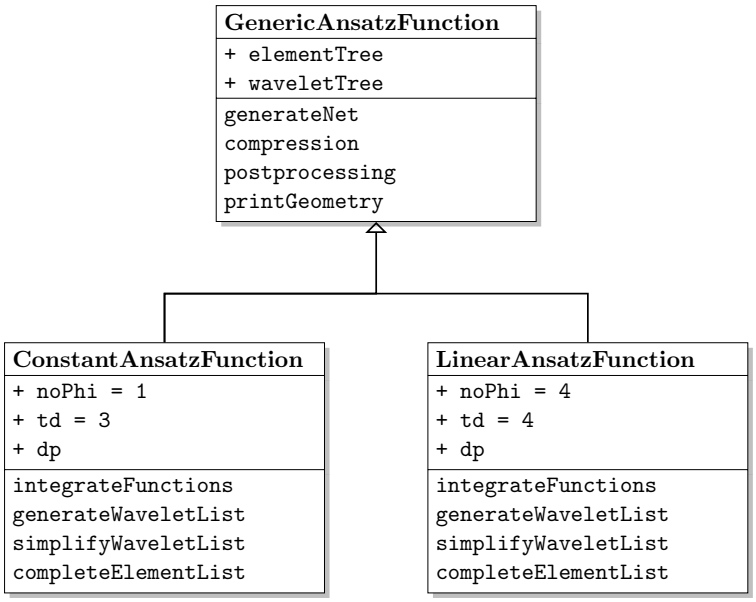


Figure 3.22. Class diagram for the wavelet solver.

CHAPTER 4

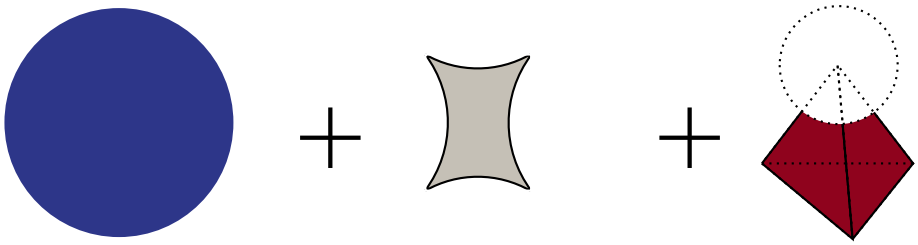
Generating the cavity

To use the BEM to solve the PCM equations, eq. (1.24) or eq. (1.27), the molecular cavity has to be discretized. In the specific case of the wavelet BEM, the cavity needs to be discretized into smooth quadrangular patches, which are uniformly refined up to a specific level n_L. The proposed algorithm is a versatile tool that can be used in most cases where a level set function or a characteristic function of the sought surface is known. Here, for the case of molecules defined by atoms, a level set function, $\chi(\boldsymbol{x})$, is constructed which returns a negative number if the point $\boldsymbol{x}$ is outside of the molecule and a positive number if it is inside the molecule. The inside of the molecule is herefore broken down into three simpler geometrical forms.

4.1. Surface decomposition

The geometry described by the atoms and the probe radius can be split up into parts stemming from three types of partial surfaces: convex molecular partial surfaces, toroidal partial surfaces, and concave spherical partial surfaces, as illustrated in

fig. 4.1. The first and simplest part of the level set function is the one coming from the atoms, shown in fig. 4.1a. A point $\boldsymbol{x}$ is inside the molecule if it is inside one of the atoms. This means that the point needs to be inside the sphere defined by the position of the nucleus and the van der Waals radius.



(a) Partial surface stemming from visible atoms of the molecule.

(b) Partial surface stemming from the interaction of two atoms.

(c) Partial surface stemming from the interaction of more than two atoms.

Figure 4.1. Surface decomposition.

The second type of partial surface arises when only two atoms interact with the probe sphere. The resulting shape is that of a sliced torus, since the probe sphere can orbit around the two atoms. This can be seen in fig. 4.1b. A more in depth illustration is found in fig. 4.2. The visible part of the torus is depicted in gray. In order to construct the level set function, one starts with the circles on the interacting spheres, denoted by `c1` and `c2`, and the orbit of the probe, `central`. The radii of the circles `c1` and `c2` are not necessarily equal. A more in depth description and a formula for calculating the level set function will follow shortly.

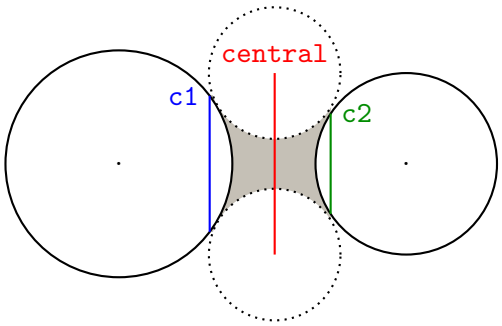


Figure 4.2. Torus detail.

The last type of partial surface comes up when more than two atoms of the molecule interact with the probe sphere at the same time, as is shown in fig. 4.1c. This happens when the probe sphere touches the molecular surface at three or more points at once. The position of the probe is thus fixed and the visible partial surface is a concave spherical partial surface defined by a partial surface of the probe sphere. In fig. 4.3, the example of three interacting atoms is depicted. The level set function for the interaction of three atoms would then return a positive number for points inside the possibly up to two tetrahedra formed by the position of the atoms and the position of the probe

sphere, but outside of the probe sphere, as depicted in fig. 4.3. This means, that for the level set function of the interaction of more than three atoms, all combinations of three interacting atoms are taken into account.

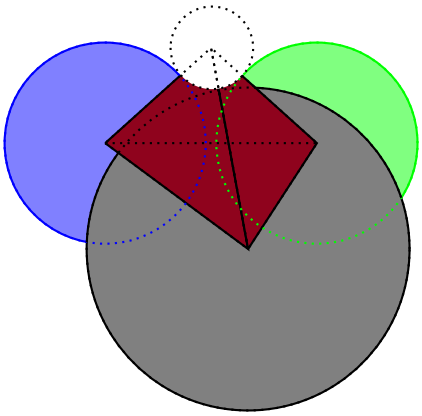


Figure 4.3. Spherical partial surface detail.

4.2. Initialization and level set function

The starting point of the algorithm is a list of atoms with their respective positions and van der Waals radii and the radius of the probe sphere, p_r. These entities are defined as instantiations of a derived class. The base class can be found in lst. 4.4, where `pos` is the position of the atom and `rad` is its radius.

Listing 4.4 SphereBase class definition.

```
class SphereBase{
    Eigen::Vector3d pos;
    double rad;
};
```

The first step is to calculate which spheres interact. To this end, an adjacency matrix is being constructed, where spheres have a *direct* interaction if the distance between the centres is smaller than the sum of their radii as given by

$$\|\texttt{si.pos} - \texttt{sj.pos}\|_2 < \texttt{si.rad} - \texttt{sj.rad}.$$

Other entries that generate an interaction are the *indirect* neighbours, where the spheres only form an interaction due to the probe radius p_r:

$$\|\texttt{si.pos} - \texttt{sj.pos}\|_2 < \texttt{si.rad} - \texttt{sj.rad} + 2p_r.$$

The complexity of constructing the adjacency matrix is $O(n_{atoms}^2)$, where n_{atoms} is the number of atoms. This matrix is being used to calculate further interactions, for example, when calculating the tori. The adjacency matrix is stored in a sparse compressed

row format and, since the interactions are symmetric, only the upper triangular part needs to be stored.

4.2.1. Intersection of two spheres

The intersection of two spheres is needed in the construction of the molecular surface, when determining the toroidal and concave spherical partial surfaces. This intersection is either a point, a circle, or nothing at all, according to the distance between the centres and the radii, as follows

$$\|\texttt{s1.pos} - \texttt{s2.pos}\|_2 \begin{cases} > \texttt{s1.rad} + \texttt{s2.rad}: & \text{no intersection,} \\ = \texttt{s1.rad} + \texttt{s2.rad}: & \text{point intersection,} \\ < \texttt{s1.rad} + \texttt{s2.rad}: & \text{circle intersection.} \end{cases}$$

Only the last case is relevant for the construction of the molecular surface since the radii of the atoms can be augmented by the probe radius and the case of the single point intersection becomes irrelevant. This case is depicted in fig. 4.5.

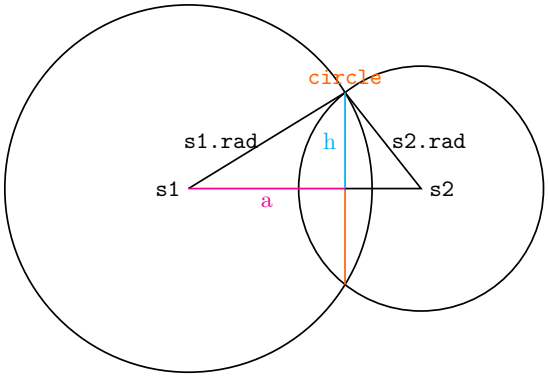


Figure 4.5. The intersection circle of two spheres.

In order to uniquely define the intersection circle `circle`, the position of the centre, the radius, and the normal of the circle plane have to be computed. The position of the centre is located on the connection line between the centres of the spheres `s1` and `s2`. The distance `a` between the sphere `s1` and the centre of the `circle` and the radius, `h`, of the `circle` satisfy the equations

$$\begin{aligned} \texttt{h}^2 + \texttt{a}^2 &= \texttt{s1.rad}^2, \\ \texttt{h}^2 + (\|\texttt{s1.pos} - \texttt{s2.pos}\|_2 - \texttt{a})^2 &= \texttt{s2.rad}^2. \end{aligned}$$

This leads to the following equations for the distance `a`

$$\texttt{a} = (\|\texttt{s1.pos} - \texttt{s2.pos}\|_2^2 + \texttt{s1.rad}^2 - \texttt{s2.rad}^2)/(2 * \|\texttt{s1.pos} - \texttt{s2.pos}\|_2) \tag{4.6}$$

and the radius `h` of the `circle`

$$\texttt{h} = \sqrt{\texttt{s1.rad}^2 - \texttt{a}^2}. \tag{4.7}$$

The position of the sought circle and the normal of the circle plane in the direction of the first sphere are thus given by

$$\begin{aligned} \texttt{circle.nrm} &= \frac{\texttt{s1.pos} - \texttt{s2.pos}}{\|\texttt{s1.pos} - \texttt{s2.pos}\|_2}, \\ \texttt{circle.pos} &= \texttt{s1.pos} - \texttt{a} \cdot \texttt{circle.nrm}. \end{aligned}$$

4.2.2. Circle line intersection

Another intersection that needs to be computed in the calculation of the molecular surface is the intersection of a circle with a line which lies in the plane of the circle.

The intersection of a circle with a line defined by the direction $\boldsymbol{d}$ and the point $\boldsymbol{p}_0$ starts by calculating the projection of the centre of the circle $\texttt{c.pos}$ onto the line

$$\texttt{c}^\perp = \boldsymbol{p}_0 + \langle \texttt{c.pos} - \boldsymbol{p}_0, \boldsymbol{d} \rangle \boldsymbol{d},$$

where $\texttt{c}^\perp$ is the projection of the point $\texttt{c.pos}$ onto the line.

Three different cases arise as depicted in fig. 4.8. The first case occurs if $\|\texttt{c.pos} - \texttt{c}^\perp\|_2 > \texttt{c.rad}$ and, in this case, the circle and the line do not intersect. The second case happens if $\|\texttt{c.pos} - \texttt{c}^\perp\|_2 = 0$ and the intersection is a single point given by $\texttt{c}^\perp$. The last case, $\|\texttt{c.pos} - \texttt{c}^\perp\|_2 < \texttt{c.rad}$, results in two intersection points for the line and the circle under consideration, $\boldsymbol{p}_-$ and $\boldsymbol{p}_+$, obtained by the equation

$$\boldsymbol{p}_\pm = \texttt{c}^\perp \pm \texttt{b} \cdot \boldsymbol{d}$$

with

$$\texttt{b} = \sqrt{\texttt{c.rad}^2 - \|\texttt{c.pos} - \texttt{c}^\perp\|_2^2}.$$

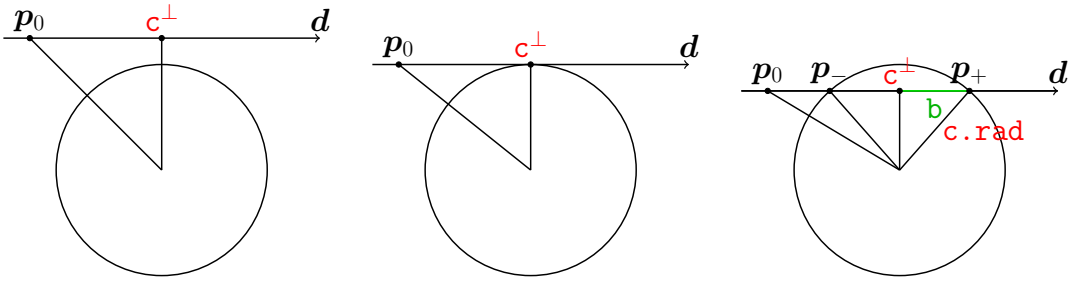


Figure 4.8. The cases for the circle-line intersection.

4.2.3. Molecular partial surface

Having the positions and the radii of the atoms at hand, the first level set function of the convex molecular partial surface, seen in fig. 4.1a and marked in blue, can easily be

defined. A point $\boldsymbol{p}$ is inside an atom if the following function returns a positive value:

$$\chi_a(\boldsymbol{p}) = \begin{cases} 1, & \text{if there exists a sphere } \texttt{si} \text{ such that } \|\boldsymbol{p} - \texttt{si.pos}\|_2 \leq \texttt{si.rad}, \\ -1, & \text{otherwise.} \end{cases}$$

4.2.4. Toroidal partial surface

The second partial surface consists of tori, as seen in fig. 4.1b and marked in gray. The first step towards defining a toroidal partial surface is the calculation of the intersection circles between the neighbours as seen in fig. 4.2. An index reference to these circles is stored in the definition of each torus. The class definition of the circle and the torus can be found in lst. 4.9.

Listing 4.9 Definitions used for the intersection of atoms.

```
class Circle:public SphereBase{
    Eigen::Vector3d nrm;
    int left, right;
    int index;
    std::vector<int> points;
};

class Torus{
    int c1, c2, central;
};
```

The construction of the circles and tori lists makes use of the adjacency matrix computed from the spheres. The complexity of computing the tori is thus $O(n)$, where n is the number of interacting atoms given by the number of non-zero entries in the adjacency matrix. For each entry in the adjacency matrix, three circles need to be computed, `c1`, `c2`, and `central`, in order to define the toroidal partial surface. The intersection between two spheres is depicted in fig. 4.5.

The three circles that need to be computed to completely define a toroidal partial surface stem from three intersections involving spheres at the positions `s1.pos` and `s2.pos`, illustrated in fig. 4.10a. The circle `c1` comes from the intersection of the sphere `s1` with radius `s1.rad` and of the sphere `s2` with the modified radius of `s2.rad`$+p_r$. The second circle, `c2`, is analogously constructed by incrementing the radius of the sphere `s1` by the probe radius and calculating the intersection. The last circle, `central`, comes from the intersection of `s1` and `s2` with the radii both augmented by the probe.

The level set function of the toroidal partial surface is not so easily written down. A graphical approach is better suited. In order to check if a point is inside a toroidal patch, the algorithm illustrated in fig. 4.10 is used. First, the point has to be between the two planes of the circles `c1` and `c2`. This can be seen in fig. 4.10a, where only the points inside the hatched gray area are still considered. The points thus belong

to the set $\{\boldsymbol{p} \in \mathbb{R}^3 : \langle \texttt{c1.nrm}, \boldsymbol{p}\rangle \cdot \langle \texttt{c2.nrm}, \boldsymbol{p}\rangle > 0\}$, where `c1.nrm` and `c2.nrm` are the normals to the circles `c1` and `c2`, which point towards the centre of the spheres that they are located on. The second test checks if the point is inside the cylinder of radius $\max\{\texttt{c1.rad}, \texttt{c2.rad}\}$, as illustrated in fig. 4.10b. To this end, the distance to the rotation axis, `s1.pos` − `s2.pos`, is needed:

$$\|\boldsymbol{d}_{2ax}\|_2^2 = \|\boldsymbol{p} - \texttt{c1.pos}\|_2^2 - \langle \boldsymbol{p} - \texttt{c1.pos}, \texttt{c1.nrm}\rangle^2.$$

If $\|\boldsymbol{d}_{2ax}\|_2 < \max\{\texttt{c1.rad}, \texttt{c2.rad}\}$, the point $\boldsymbol{p}$ is inside the cylinder between the planes of the circles, `c1` and `c2`, and radius $\max\{\texttt{c1.rad}, \texttt{c2.rad}\}$ depicted in hatched gray in fig. 4.10b. The next step uses `a`, the distance between the sphere `s1` and the circle `central`, given by eq. (4.7), and the radius `central.rad`, defined in eq. (4.7) as `h`.

If $\|\boldsymbol{d}_{2ax}\|_2 > 0$, the point $\boldsymbol{p}$ and the two circle centres, `c1.pos`, `c2.pos`, form a plane. In this plane, the position of the two probe sphere centres $\boldsymbol{o}_\pm$ where the probe sphere touches the spheres `s1` and `s2`, as depicted in fig. 4.10e are computed:

$$\boldsymbol{o}_\pm = \texttt{central.pos} \pm \texttt{h} \cdot \boldsymbol{d}_{2ax}/\|\boldsymbol{d}_{2ax}\|_2.$$

Here, $\boldsymbol{d}_{2ax}$ is given by

$$\boldsymbol{d}_{2ax} = \boldsymbol{p} - \texttt{c1.pos} - \langle \boldsymbol{p} - \texttt{c1.pos}, \texttt{c1.nrm}\rangle \texttt{c1.nrm}.$$

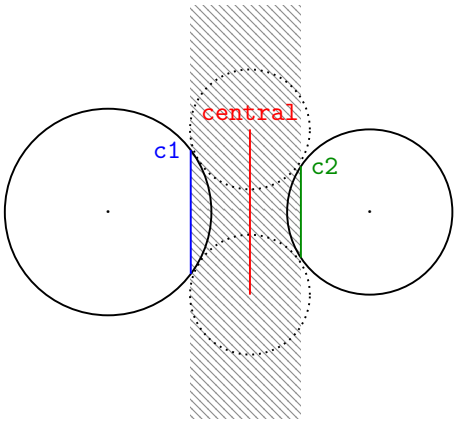
We will denote by $\boldsymbol{o}_+$ and $\boldsymbol{o}_-$ the upper and lower centre coordinates of the position of the probe, respectively. The point $\boldsymbol{p}$ is inside the cavity if $\|\boldsymbol{p} - \boldsymbol{o}_+\|_2 > p_r$ and $\|\boldsymbol{p} - \boldsymbol{o}_-\|_2 > p_r$, meaning that the points that lie inside the probe spheres are being discarded.

If $\|\boldsymbol{d}_{2ax}\|_2 \approx 0$, the point $\boldsymbol{p}$ is almost collinear with the circle centres `c1.pos` and `c2.pos`. This means that there is no well defined plane in which the positions, $\boldsymbol{o}_\pm$, of the probe sphere can be computed. If $\texttt{h} - p_r > 0$, then $\boldsymbol{p}$ is inside the molecule, as illustrated in light gray in fig. 4.10c. If $\texttt{h} - p_r < 0$, then the probe sphere intersects the connection line and the point, $\boldsymbol{p}$, needs to be outside of the probe sphere. The resulting surface is the spindle case seen in fig. 4.10e. The intersection points of the probe sphere and the connection line are calculated, resulting in a segmentation of the connection line in a sequence of points $[\texttt{c1.pos}, \boldsymbol{c}_-, \boldsymbol{c}_+, \texttt{c2.pos}]$. All the test points that lie in the segments $[\texttt{c1.pos}, \boldsymbol{c}_-]$ and $[\boldsymbol{c}_+, \texttt{c2.pos}]$ are marked as inside the torus.

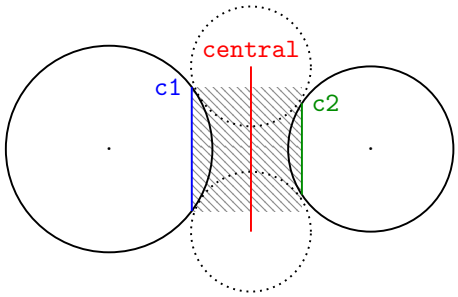
If the point is inside one of the toroidal partial surfaces, the algorithm returns the value 2 and is marked gray in the figures.

4.2.5. Intersection of three spheres or more

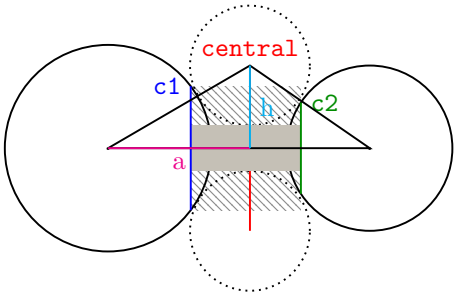
Whenever two or more `central` circles from different tori intersect, a concave spherical patch has to be generated, as illustrated in fig. 4.3. The intersection of two circles in three dimensions can be one or two points, the entire circle, or nothing at all. Since, the circles are distinct, we are interested in the case of point intersections with one or two points. These intersection points are stored in a vector of `CirclePoints` with the



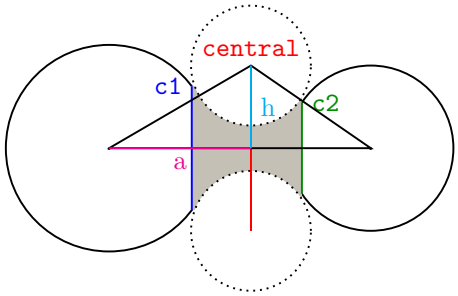
(a) Only points between the planes of the circles are considered.



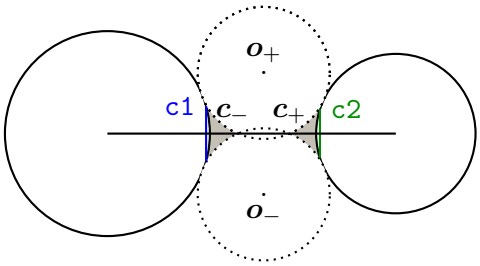
(b) Torus points need to be in the cylinder of radius $\max\{\texttt{c1.rad}, \texttt{c2.rad}\}$.



(c) If $\texttt{h} - p_r > 0$, all points in the cylinder of radius $\texttt{h} - p_r$ are in the molecule.



(d) Remaining points should be outside the probe sphere.



(e) If $\texttt{h} - p_r < 0$ and the point lies on the connecting axis, the point should not be between $\boldsymbol{c}_-$ and $\boldsymbol{c}_+$.

Figure 4.10. Algorithm for detecting points inside a toroidal patch.

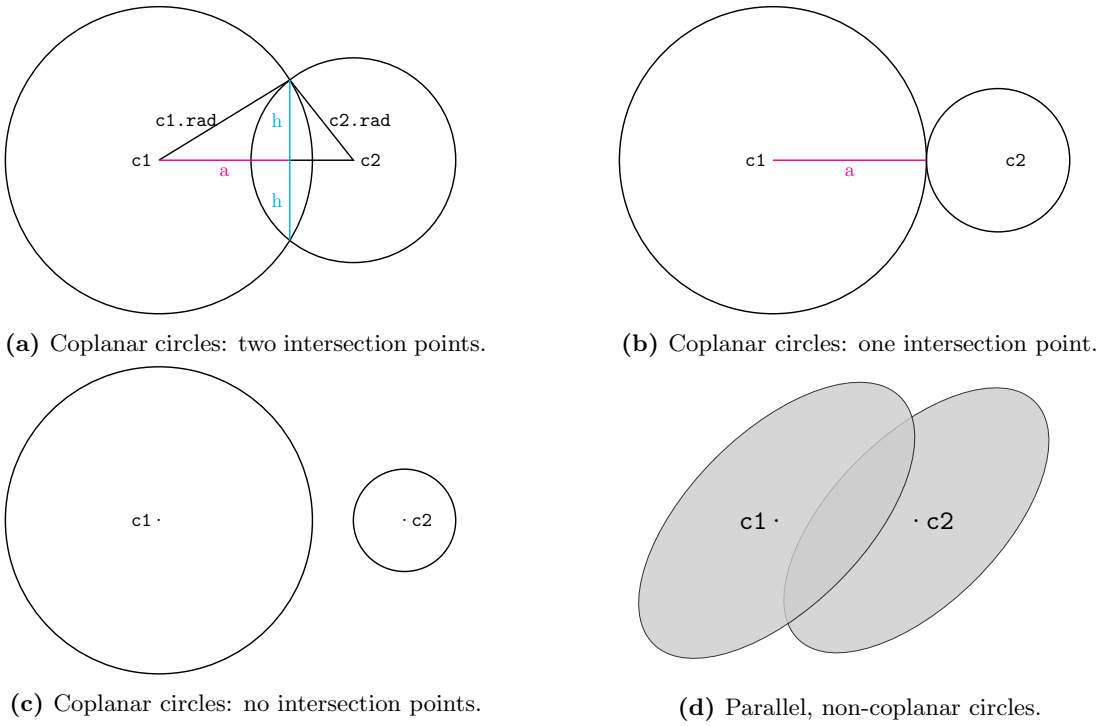
definition found in lst. 4.11. The case of concentric circles or identical circles cannot appear in our situation. Concentric circles would only appear if two atoms would be entirely enclosed in another set of two atoms. The second situation can happen if again fully enclosed atoms exist in the system or if two atoms appear two times each.

Listing 4.11 Definitions used for the intersection of circles on atoms.

```
class CirclePoints{
    Eigen::Vector3d pos;
    std::vector<int> circles, spheres;

    int addCircles(int index);
    int addSphere(int index);
};
```

Two relevant cases arise when calculating the intersection between two circles, `c1` and `c2`, in three dimensions. The case of circles residing in parallel planes is one of them, but it has a contribution only if the circles are in the same plane. The cases for parallel circles are depicted in fig. 4.12. Hence, eq. (4.6) and eq. (4.7) can be used to calculate the possibly up to two intersection points.



(a) Coplanar circles: two intersection points.

(b) Coplanar circles: one intersection point.

(c) Coplanar circles: no intersection points.

(d) Parallel, non-coplanar circles.

Figure 4.12. Possible intersections between parallel circles.

The second relevant case appears when the circles belong to intersecting planes, as seen in fig 4.13. In this situation, the first step is to check if the spheres defined

by (c1.pos, c1.rad) and (c2.pos, c2.rad), respectively, are intersecting. In case the spheres do not intersect, also the circles cannot intersect. Otherwise, the common line of the two planes of the central circles, c1 and c2, is found. The direction of the line is given by the cross product $\boldsymbol{d} = \texttt{c1.nrm} \times \texttt{c2.nrm}$, as seen in fig. 4.13a. In order to completely define the line, a point on the line has to be determined. In case the z-component of the direction of the line is non-zero, $\boldsymbol{d}_z \neq 0$, the point $\boldsymbol{p}_0$ given by

$$\begin{aligned}
\boldsymbol{p}_{0x} &= \frac{\texttt{c2.nrm}_y\langle\texttt{c1.nrm}, \texttt{c1.pos}\rangle - \texttt{c1.nrm}_y\langle\texttt{c2.nrm}, \texttt{c2.pos}\rangle}{\boldsymbol{d}_z}, \\
\boldsymbol{p}_{0y} &= \frac{\texttt{c1.nrm}_x\langle\texttt{c2.nrm}, \texttt{c2.pos}\rangle - \texttt{c2.nrm}_x\langle\texttt{c1.nrm}, \texttt{c1.pos}\rangle}{\boldsymbol{d}_z}, \\
\boldsymbol{p}_{0z} &= 0,
\end{aligned}$$

belongs to the common line. Analogous equations can be found for the case of $\boldsymbol{d}_x \neq 0$ or $\boldsymbol{d}_y \neq 0$. Since the normals, c1.nrm and c2.nrm, are not parallel, at least one of the entries is different to zero and a point can be determined. The intersection of the line with each circle c1 and c2 is computed, as described earlier. The possible cases are depicted in fig. 4.13. The equal intersection points are stored as a component of the CirclePoints vector. Note that the interaction of at least three atoms results in up to two CirclePoints, defined by the positions where the probe sphere remains fixed.

The points inside the molecule are also inside the tetrahedron formed by the three sphere atoms interacting and the position of the probe, but outside of the probe sphere. To check if a point $\boldsymbol{p}$ is inside the tetrahedron, one has to solve the system of linear equations

$$\begin{pmatrix} \texttt{s1.pos} \\ \texttt{s2.pos} \\ \texttt{s3.pos} \\ \texttt{pr.pos} \end{pmatrix} \boldsymbol{x} = \boldsymbol{p}.$$

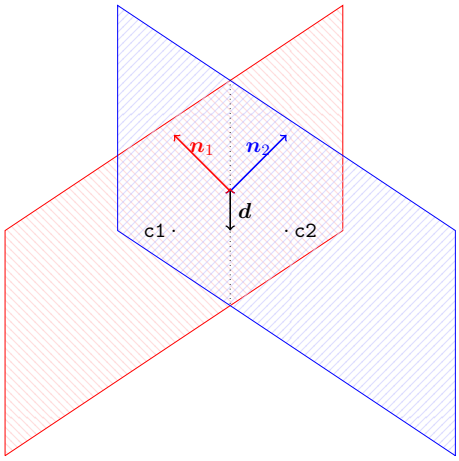
The solution $\boldsymbol{x}$ represents the barycentric coordinates of the point $\boldsymbol{p}$ in the tetrahedron formed by the positions s1.pos, s2.pos, s3.pos, and pr.pos. If all the entries are between 0 and 1, the point $\boldsymbol{p}$ is inside the tetrahedron. If the distance to the position of the probe sphere is larger than the radius of the probe, $\|\texttt{pr.pos} - \boldsymbol{p}\|_2 > p_r$, the point is inside the molecule and the algorithm returns the value 3 and marks the surface in red.

If a point is in none of the partial surfaces, the point under consideration is outside of the molecule.

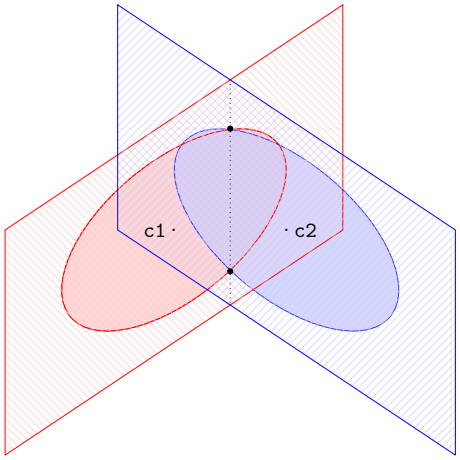
4.2.6. Domain decomposition

Testing for each point whether or not it belongs to any of the partial surfaces involved might take a long time, which is why a domain decomposition technique is used to make the search run in $O(1)$. During the initialization phase, the bounding box surrounding the molecule is split up into cubes of size

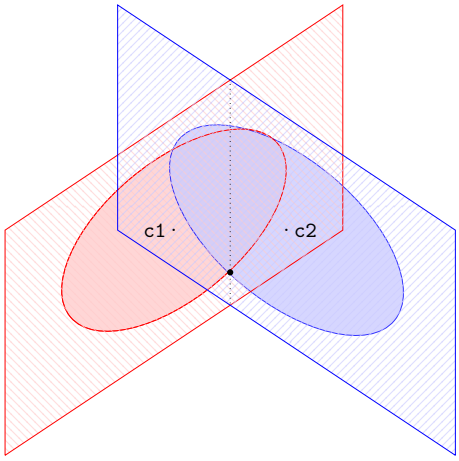
```
step_size = min{si.rad,pr}/geometryFactor,
```
(4.14)



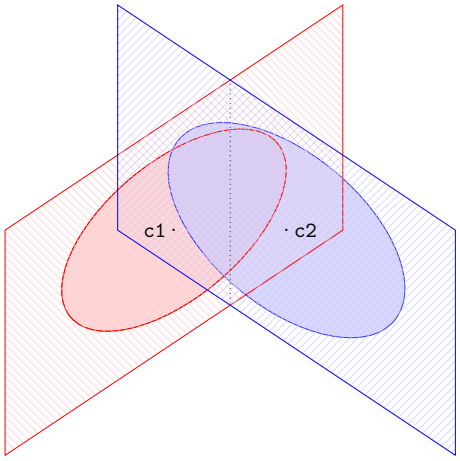
(a) Intersection of two planes.



(b) Non-coplanar circles: two intersection points.



(c) Non-coplanar circles: one intersection point.



(d) Non-coplanar circles: no intersection points.

Figure 4.13. Possible intersections between non-coplanar, non-parallel circles.

where `geometryFactor` is a parameter that can be set. For each cube, an index to the relevant partial surfaces is stored. A point $\boldsymbol{p}$, that needs to be tested, is only to be tested against the partial surfaces relevant for the associated cube.

The domain decomposition itself can be improved by using the information of the relevant atoms in determining whether or not a toroidal or spherical surface is important. This way, the search for the more complex partial surfaces is simplified. An atom is considered to be relevant if the midpoint of the cube is in the neighbourhood of the atom. The neighbourhood is defined by a sphere at the same position with the radius augmented by the radius of the probe sphere and the cube size. If the subsequent more complex geometries contain at least one relevant atom, they need to be tested, otherwise the cube is not within the volume covered by that partial surface.

After the initialization phase, the information of the cube domain decomposition can be used to find the relevant partial geometries in constant time for all points that will be generated.

4.3. Initial triangulation

The starting point of the initial triangulation contains additionally to the input of the program also the points and circles resulting from sphere-sphere interactions, circle-circle interactions, the tori, the `CirclePoints` and the domain decomposition. This information is used to find out if a point, $\boldsymbol{p}$ is inside the geometry or not. The first step is to generate a fine approximation of the surface, by using the marching cubes algorithm. This initial mesh has to guarantee that the fine structures of the geometry are resolved and that quadrangular patches can be formed.

4.3.1. Marching cubes

The initial triangulation is obtained from a marching cube algorithm, as originally described in [46], applied inside the bounding box of the molecule. The bounding box of the molecule is defined as the cuboid that fully encloses the molecule. The lengths of the sides are given as a function over all atoms `s` of the molecule as follows:

$$
\begin{aligned}
l_x &= \max_{\mathtt{s}} \left\{ \mathtt{s.pos}_x + \mathtt{s.rad} + p_r \right\} - \min_{\mathtt{s}} \left\{ \mathtt{s.pos}_x - \mathtt{s.rad} - p_r \right\}, \\
l_y &= \max_{\mathtt{s}} \left\{ \mathtt{s.pos}_y + \mathtt{s.rad} + p_r \right\} - \min_{\mathtt{s}} \left\{ \mathtt{s.pos}_y - \mathtt{s.rad} - p_r \right\}, \\
l_z &= \max_{\mathtt{s}} \left\{ \mathtt{s.pos}_z + \mathtt{s.rad} + p_r \right\} - \min_{\mathtt{s}} \left\{ \mathtt{s.pos}_z - \mathtt{s.rad} - p_r \right\}.
\end{aligned}
$$

The molecule is translated such that the origin of the cuboid lies in the origin of the coordinate system. This is done by transporting each sphere centre by

$$
\begin{aligned}
x_{\min} &= \min_{\mathtt{s}} \left\{ \mathtt{s.pos}_x - \mathtt{s.rad} - p_r \right\}, \\
y_{\min} &= \min_{\mathtt{s}} \left\{ \mathtt{s.pos}_y - \mathtt{s.rad} - p_r \right\}, \\
z_{\min} &= \min_{\mathtt{s}} \left\{ \mathtt{s.pos}_z - \mathtt{s.rad} - p_r \right\}.
\end{aligned}
$$

Uniformly distributed gridpoints are generated with the stepsize given as a function of the radii of the atoms and the probe radius, like in eq. (4.14). The gridpoints can be separated by means of the level set functions defined earlier into inner points and outer points. The cubes which have corner points both, outside and inside the molecule, are the cubes which contain the surface. There are several cases of boundary cubes which are summarized in fig. 4.15, depending on the way that the surface cuts the cube. The points belonging to the same group, inside or outside the cavity, are marked with the same color, blue or green. If the points belonging to the same group do not form a connected graph with respect to the edges of the cube, the refinement was not chosen fine enough and the stepsize for the cubes has to be adjusted. The complexity of classifying the cubes into exterior, interior and boundary cubes is of order $O(n^3)$.

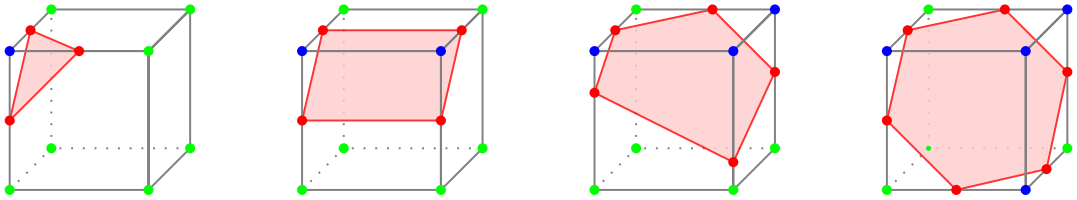


Figure 4.15. Intersection of the surface with the cubic grid.

By using the bisection algorithm shown in algo. 4.16, the surface points, marked in red in fig. 4.15, can be found up to a given precision.

Algorithm 4.16 Bisection algorithm for finding the surface.

```
procedure BISECTION(P, Q)        ▷ find point on surface between points P and Q
    if ‖P − Q‖ < ε_surf then
        return P                                   ▷ points are close enough
    let M = 0.5(P + Q)                             ▷ find midpoint
    if M on the same side as P then
        return bisection(M, Q)
    else
        return bisection(P, M)
```

The resulting surface description is an unstructured grid stemming from the intersections, as seen in fig. 4.15. The elements of the unstructured grid can potentially be very small or have very small edges, as seen in fig. 4.17.

4.3.2. First triangular mesh

In the next step, non-triangular elements of the surface grid are subdivided into triangles by recursively inserting a new edge with the shortest length. This leads to a triangular mesh with possibly ugly triangles. Nonetheless, since the elements are convex, the division into triangles can be generated without any further tests.

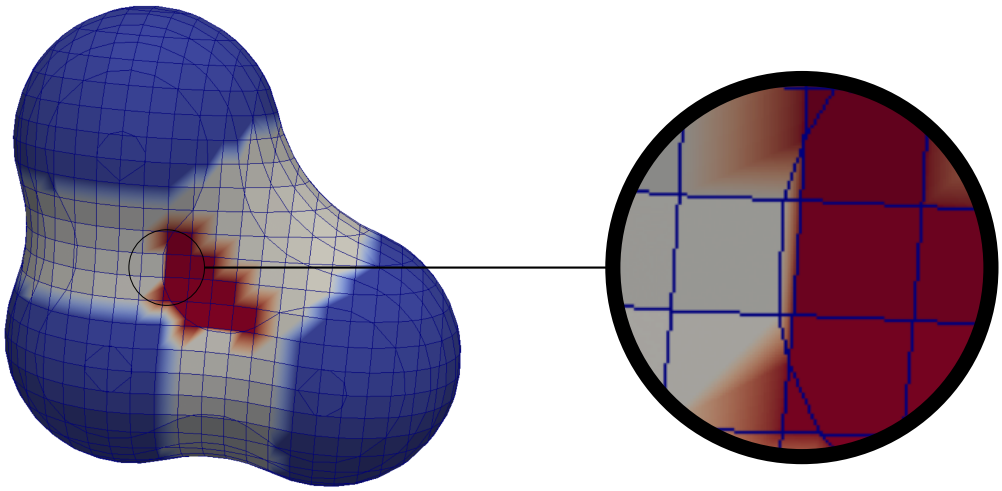


Figure 4.17. Unstructured grid resulting from the marching cubes algorithm.

For the triangle Δ with edges e_1, e_2, and e_3, we associate a fitness measure defined by

$$fit(\Delta) = \frac{4\sqrt{3}|\Delta|}{\sum_i |e_i|^2}, \tag{4.18}$$

where $|e_i|$ denotes the length of the edge e_i in Δ. This fitness measure takes values between 0 and 1. An equilateral triangle would have a fitness of 1. The further the triangle deviates from being equilateral, the smaller the value of the fitness becomes. This definition takes only the angles, α, β, and γ, of the triangle, Δ, into account and can be rewritten as

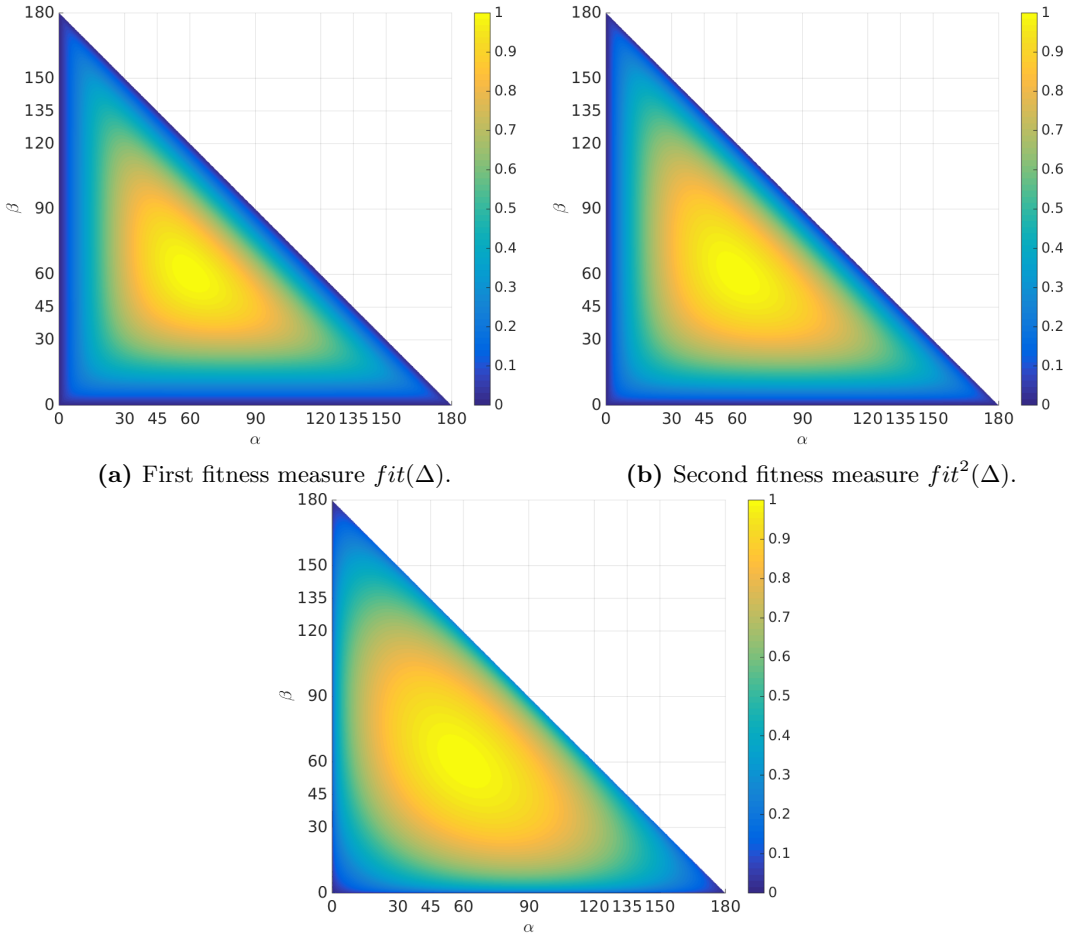
$$fit(\Delta) = \frac{2\sqrt{3}\sin(\alpha)\sin(\beta)\sin(\gamma)}{\sin^2(\alpha)+\sin^2(\beta)+\sin^2(\gamma)}.$$

Different definitions of the fitness measure can also be used, for example

$$\begin{aligned} fit^2(\Delta) &= \frac{6\sqrt{3}\sin(\alpha)\sin(\beta)\sin(\gamma)}{(\sin(\alpha)+\sin(\beta)+\sin(\gamma))^2} = \frac{12\sqrt{3}|\Delta|}{(|e_1|+|e_2|+|e_3|)^2}, \\ fit^3(\Delta) &= \frac{2\sqrt{3}\sqrt[3]{\sin(\alpha)\sin(\beta)\sin(\gamma)}}{3} = \frac{4\sqrt{3}|\Delta|}{\sqrt[3]{(|e_1|\ |e_2|\ |e_3|)^2}}. \end{aligned} \tag{4.19}$$

These measures also have values between 0 and 1, with 1 for the equilateral case. A comparison plot can be found in fig. 4.20, where two angles of the triangle vary according to $\alpha+\beta \leq \pi$ and the third is computed by $\gamma = \pi - \alpha - \beta$. This figure shows that all measures have a maximum for the equilateral triangle and go to zero the more the triangle degenerates. The measure (4.18) chosen here decays the fastest.

The first triangular mesh improvement step replaces edges in an ascending order of the replacement error. This coarsening stops when the fitness level of the worst triangle reaches a certain threshold. The algorithm for replacing an edge is presented in algo. 4.22.



(a) First fitness measure $fit(\Delta)$.

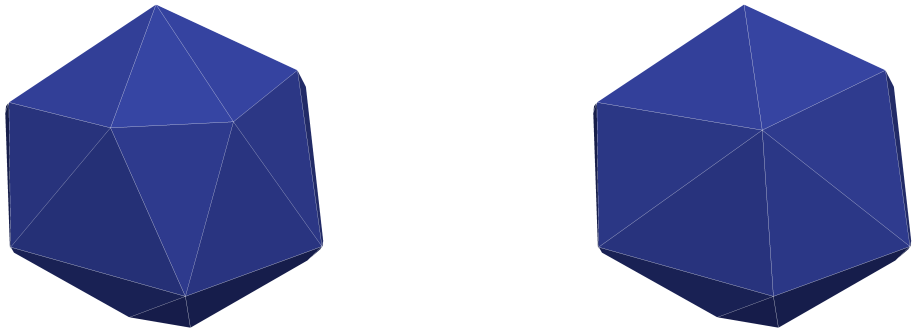
(b) Second fitness measure $fit^2(\Delta)$.

(c) Third fitness measure $fit^3(\Delta)$.

Figure 4.20. Comparison of fitness measures for triangles as described in eq. (4.18) and eq. (4.19).

Algorithm 4.22 Replacing an edge with a point.

procedure REPLACEEDGE($mesh, e_i$) ▷ replace edge with index e_i in mesh
  let $\texttt{mesh}\rightarrow\texttt{edgeNodes}[e_i] = (n_1, n_2)$ ▷ node index of the edge
  let $\texttt{mesh}\rightarrow\texttt{edgeElements}[e_i] = (e_1, e_2)$ ▷ elements of the edge
  let $\texttt{mesh}\rightarrow\texttt{nodes}[n_i] = \boldsymbol{q}_i$, $i = 1, 2$ ▷ vertices of the edge
  find $elements = \{\Delta \in \texttt{mesh} : n_1 \in \Delta \text{ or } n_2 \in \Delta\}$
  calculate $\boldsymbol{A} \in \mathbb{R}^{|elements|\times 3}$, $\boldsymbol{A}_{i,:} = \boldsymbol{n}_{\Delta_i}$, $\Delta_i \in elements$
  calculate $\boldsymbol{b} \in \mathbb{R}^{|elements|}$, $\boldsymbol{b}_i = \langle \boldsymbol{n}_{\Delta_i}, \boldsymbol{q}_i \rangle$, $\Delta_i \in elements$, $\boldsymbol{q}_i \in \Delta_i$
  calculate via QR decomposition $\boldsymbol{p}_{\text{new}} \approx \boldsymbol{A}^{-1}\boldsymbol{b}$
  for each $\Delta_j = \Delta(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) \in elements$ **do** ▷ compatibility condition
    $\boldsymbol{d} = \boldsymbol{p}_1 - \boldsymbol{p}_2$, with the point $\boldsymbol{p}_3$ on the edge e_i
    $\boldsymbol{n} = \boldsymbol{d} \times \boldsymbol{n}_{\Delta_j}$
    check that $\langle \boldsymbol{n}, \boldsymbol{p}_3 - \boldsymbol{p}_1 \rangle \cdot \langle \boldsymbol{n}, \boldsymbol{p}_{\text{new}} - \boldsymbol{p}_1 \rangle > 0$ ▷ points $\boldsymbol{p}_{\text{new}}$ and $\boldsymbol{p}_3$ on the same side of plane defined by $\boldsymbol{d}$ and $\boldsymbol{n}_{\Delta_j}$



(a) Situation before edge replacement.

(b) Situation after edge replacement.

Figure 4.21. Edge replacement by a point.

It takes the mesh description and the index of the edge, $e = (\boldsymbol{q}_1, \boldsymbol{q}_2)$, to be replaced as input and gives the new point, $\boldsymbol{p}_{\text{new}}$, and the error, $error(e)$, made by the replacement as an output. The point $\boldsymbol{p}_{\text{new}}$ is chosen as the least squares solution of the following problem:

$$\boldsymbol{p}_{\text{new}} = \arg\min_{\boldsymbol{q}\in\mathbb{R}^3} \|\boldsymbol{A}\boldsymbol{q} - \boldsymbol{b}\|_2. \tag{4.23}$$

Here, the matrix $\boldsymbol{A}$ is given by $\boldsymbol{n}_i$, the normals to the triangles containing either $\boldsymbol{q}_1$ or $\boldsymbol{q}_2$, and $\boldsymbol{b} = \langle \boldsymbol{n}_i, \boldsymbol{p}_i \rangle$, where $\boldsymbol{p}_i$ is a point in the triangle corresponding to the index i of the system matrix. In order to get a better understanding of the entries of the matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$, let us consider the situation found in fig. 4.24. The matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$ are here given by

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{n}_1^T \\ \vdots \\ \boldsymbol{n}_4^T \\ \boldsymbol{n}_5^T \\ \vdots \\ \boldsymbol{n}_9^T \end{pmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{pmatrix} \langle \boldsymbol{n}_1, \boldsymbol{q}_2 \rangle \\ \vdots \\ \langle \boldsymbol{n}_4, \boldsymbol{q}_2 \rangle \\ \langle \boldsymbol{n}_5, \boldsymbol{q}_1 \rangle \\ \vdots \\ \langle \boldsymbol{n}_9, \boldsymbol{q}_1 \rangle \end{pmatrix}.$$

The new point is calculated by a QR decomposition of the system matrix $\boldsymbol{A}$ from eq. (4.23). The norm of the residuum of the solution is used to decide upon which edge to replace first. The error for the edge marked with magenta is thus given by

$$error(e) = \|\boldsymbol{A}\boldsymbol{p}_{\text{new}} - \boldsymbol{b}\|_2.$$

The system represents the plane equations of all triangles that are involved in the replacement of the edge by $\boldsymbol{p}_{\text{new}}$. The perfect point would thus lie in all the planes involving $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. The edge that will be chosen is the one that has the smallest error.

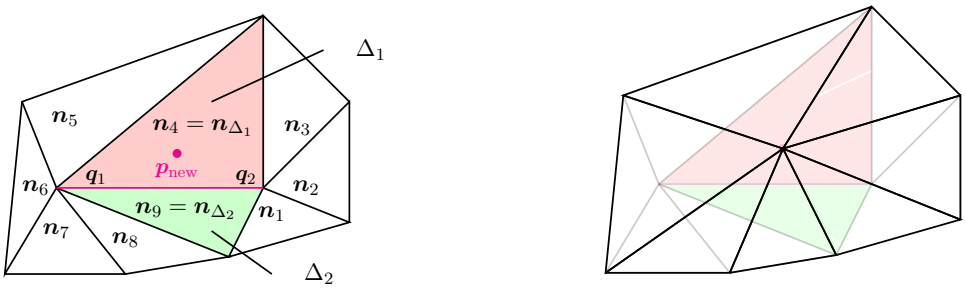


Figure 4.24. Example of edge replacement.

The chosen point is then projected onto the surface in the direction of the mean of the normals of the two triangles, Δ_1 and Δ_2, containing the edge to be replaced:

$$\boldsymbol{n}_{1/2} = \frac{1}{2}(n_{\Delta_1} + n_{\Delta_2}). \tag{4.25}$$

Before the edge is replaced, the point has to pass an admissibility test to guarantee that the new surface does not have self-intersections. Namely, we have to check whether the new point is on the same side of a plane as the old point for each triangle still contained in the new mesh. Let the old triangle be $\Delta_{old} = \Delta(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ with the normal $\boldsymbol{n}_\Delta$ and the new triangle $\Delta_{new} = \Delta(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_{\text{new}})$, where $\boldsymbol{p}_3$ is either $\boldsymbol{q}_1$ or $\boldsymbol{q}_2$. The admissibility check ensures that the following inequality holds:

$$\langle (\boldsymbol{p}_1 - \boldsymbol{p}_2) \times \boldsymbol{n}_\Delta, \boldsymbol{p}_{\text{new}} - \boldsymbol{p}_1 \rangle \langle (\boldsymbol{p}_1 - \boldsymbol{p}_2) \times \boldsymbol{n}_\Delta, \boldsymbol{p}_3 - \boldsymbol{p}_1 \rangle > 0. \tag{4.26}$$

The test looks at the plane perpendicular to each of the old triangles, Δ_{old}, which contain both, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, marked in blue in fig. 4.27. If the new point is in the hatched side of the blue plane it is admissible, otherwise it is discarded. This plane is characterized by the normal given by $\boldsymbol{n} = (\boldsymbol{p}_1 - \boldsymbol{p}_2) \times \boldsymbol{n}_\Delta$. If there is at least one triangle where the inequality (4.26) does not hold, the replacement would result in a self-intersection of the surface and the edge is not replaced, independent of the error being made.

A simple case, where such a self-intersection occurs, is illustrated in fig. 4.28. The triangles are all in one plane. In the left subfigure, the situation before the replacement is depicted. Replacing the edge in the centre, highlighted in magenta, for example by its midpoint, results in a self-intersecting surface involving the marked green, red, and blue triangles.

The complexity of the mesh improvement by the coarsening algorithm is governed by the complexity of calculating the sorted list of errors for the edge replacement. A naive implementation would use a standard vector and the quick-sort algorithm which would result in a complexity of $O(n_e \log(n_e))$, with n_e being the number of edges. The drawback consists in the fact that the vector needs to be computed anew every time an edge is replaced. The method used to improve the complexity in the present implementation is the usage of a doubly linked list instead of a vector. The doubly linked list is initialized from a vector, built as previously described, and kept up to date

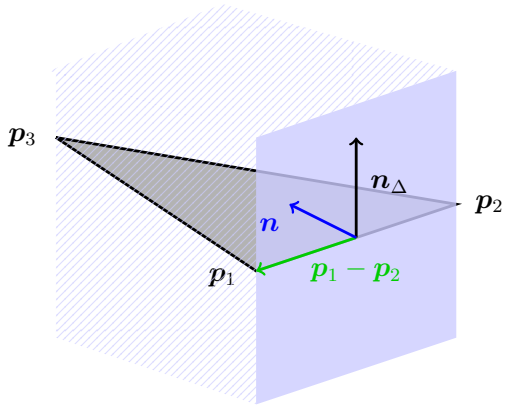


Figure 4.27. Example of admissibility test.

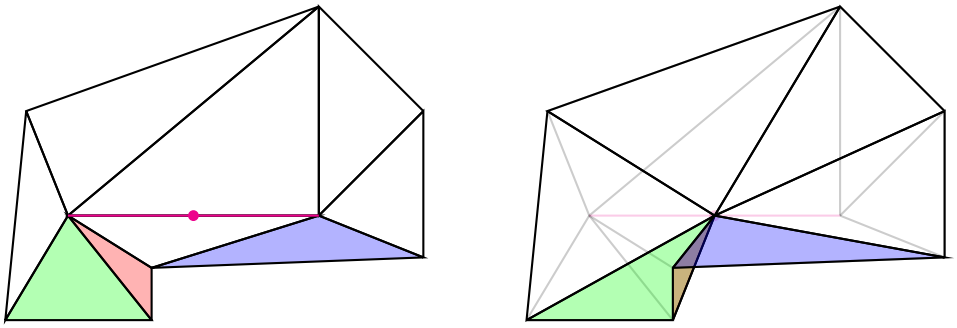


Figure 4.28. Example of a self-intersecting surface after an edge replacement. The triangles that form a problem are highlighted in green, red, and blue.

Listing 4.29 Node structure of the doubly linked list.

```
struct node{
  int edgeIdx;
  double val;
  Eigen::Vector3d point;
  node* next;
  node* prev;
};
```

with every replacement. The nodes of the doubly linked list contain the information found in lst. 4.29.

A reverse access vector of pointers is also employed in order to easily find the nodes of specific edges. A schematic of the structure can be seen in fig. 4.30, where for example the edge with the index 1 has the smallest error of replacement and the largest error belongs to the edge with the largest index, $n_e - 1$.

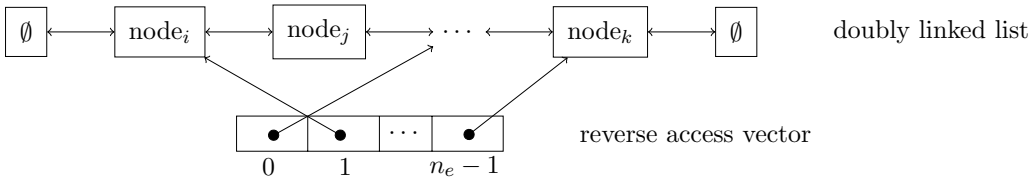


Figure 4.30. Structure of the doubly linked list and the reverse access vector.

The replacement of the magenta edge from fig. 4.31 will be taken as reference to exemplify the changes that need to be done to the mesh and the doubly linked list. The edges marked magenta and dashed black need to be removed from the mesh. Likewise, the triangles coloured lightgray are deleted as well as one of the nodes of the magenta edge. The remaining node of the magenta edge changes the coordinates to the new point, $\boldsymbol{p}_{\text{new}}$, calculated by algo. 4.22. The blue edges will need to update their node information. The orange triangles are altered as well, since their nodes, edges, area, normal, and fitness will change. Finally, the doubly linked list needs to be updated for both the green and the blue edges.

The update of the linked list is performed in two steps, the reverse access list is used to remove the edge information from the list in $O(1)$. Calling the algo. 4.22 for all the edges that need to be updated gives the new information that needs to be added to the list. One addition has complexity $O(n_e)$, where n_e is the current size of the list. This leads to the overall complexity $O(n_{update} n_e)$, where n_{update} depends on the degree of the nodes in the direct neighbourhood of the edge being replaced. This implementation performs much faster compared to the naive implementation when the number of edges in the mesh is very large and has similar computation times for small meshes.

4.3.3. Even number of triangles

Since the goal is to create quadrangular patches on the surface, the next step ensures an even number of triangles in the surface grid. Several methods can be employed to discretize one triangle into an even number of triangles, as seen in fig. 4.32. After the first improvement step, described in the previous section, the remaining triangles should be closer to equilateral triangles, such that the first refinement method of fig. 4.32 is used. This keeps the overall quality of the mesh the same and introduces points only on the edges of the elements, which is of linear complexity in the number of edges, $O(n_e)$. The third refinement of fig. 4.32 could be used to decrease the size of the longest edges. The implementation, however, can be tricky, since it does not symmetrically refine all edges and the best choice of edges to be refined has to be found. A combination of

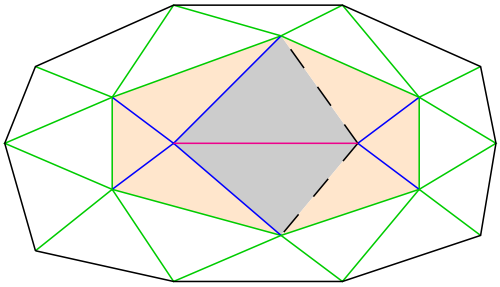
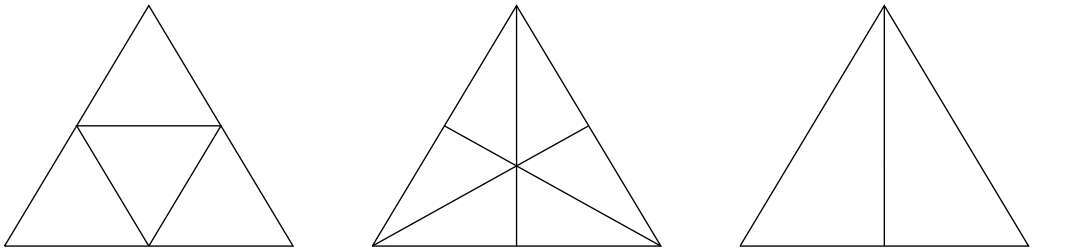


Figure 4.31. Edges that need to be updated by replacing the edge marked in magenta. The blue edges are directly affected by the edge replacement. One of the nodes will be updated to be the new point. The triangles in orange need to update area, normal, fitness, indices of nodes and edges. The green edges are affected by the change of the normals of the orange triangles and the edge replacement algorithm needs to be evaluated again.



(a) Refinement of one triangle into four triangles.

(b) Refinement of one triangle into six triangles.

(c) Refinement of one triangle into two triangles.

Figure 4.32. Refinement methods for triangles.

different refinement methods can also be used. This would result in a refinement of one triangle in twelve new triangles if for example the first two methods are employed. Since the first mesh improvement guarantees triangles with high fitness, the first refinement into four triangles is applied.

4.3.4. Final improvement

The refinement and projection onto the surface can lead to a few triangles that have a fitness under the given threshold, so the new mesh is again improved until the threshold in the fitness of the elements is reached again. This eliminates mainly very small elements that have been created due to the refinement. This elimination is done via the algorithm found in algo. 4.22. Although edges are being replaced by points, it does not change the fact that we now have an even number of triangles, since each replacement removes exactly two triangles from the mesh.

All steps described so far are depicted in fig. 4.33 for a toy example. On the first line, the initial configuration of the interlocking spheres and the unstructured grid are shown. On the second line, the initial triangulation obtained from splitting the unstructured grid and the optimized initial triangulation are depicted. The final triangulation, seen in fig. 4.33d, would enable the use of boundary element methods which are based on the standard surface representation by planar triangles.

4.4. Generating patches

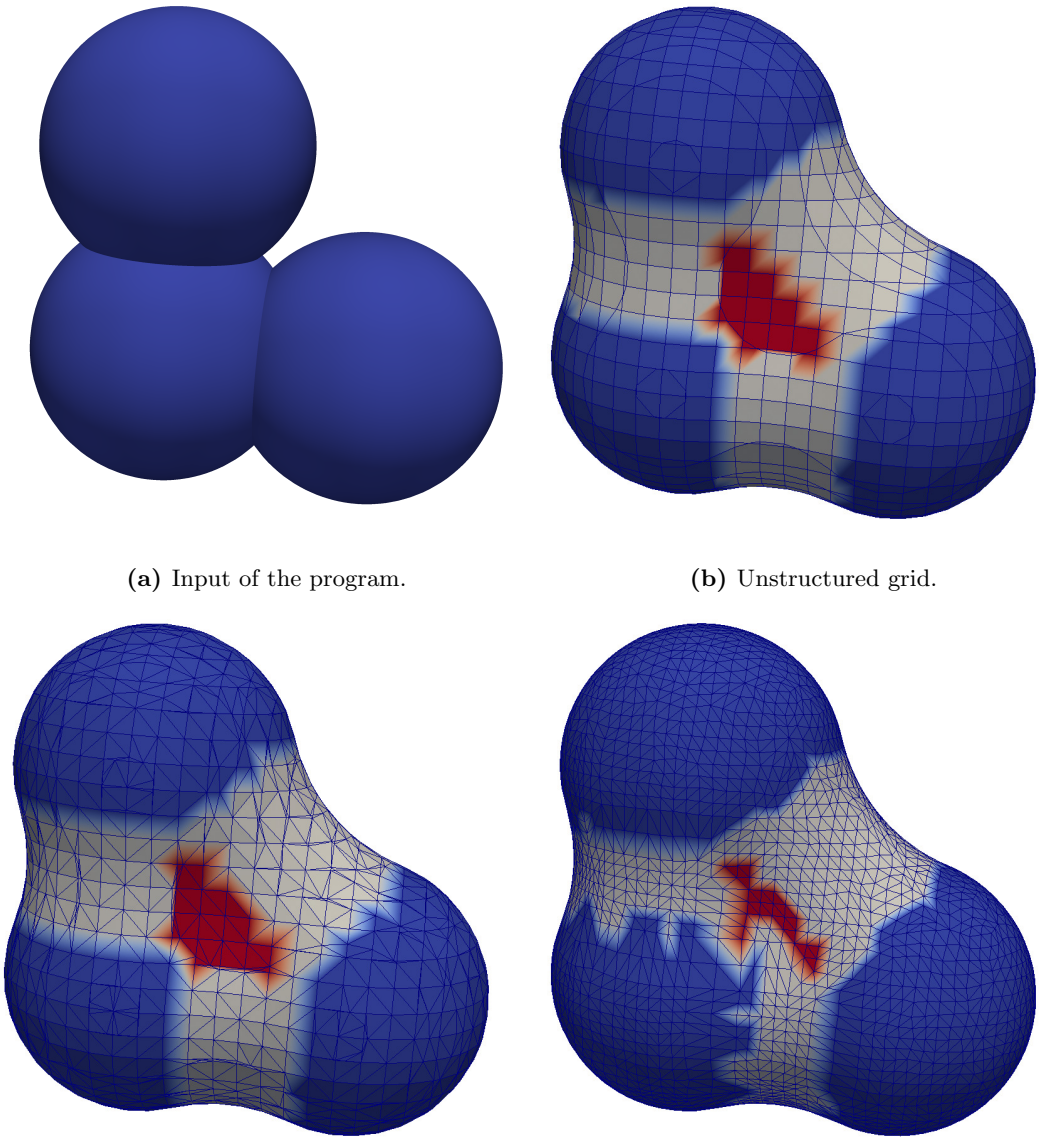
In order to generate the subdivision of the surface into rectangular patches, the same algorithm for replacing edges by points is used, as in the previous section, namely algo. 4.22. This is done until a given number of triangles is left in the surface description, $n_\Delta = 2n_\square$. The desired final number of patches is determined by the input parameter $n_\square$. The resulting coarse geometry can be seen in fig. 4.36a. A graph based algorithm is used to calculate a minimum-cost matching that reflects which triangles should be put together.

4.4.1. Convexity in three dimensions

Not any two triangles can be merged into a patch. An obvious requirement for a patch in a plane would be the fact that it should be convex. The concept of convexity will be extended to three dimensions in the context of generating patches for wavelet BEM.

4.4.1.1. Convexity from plane projection

Convexity of a quadrangle can be claimed in a plane if the diagonals are inside the quadrangle, as represented in fig. 4.34a. For a quadrangle given by its four vertices $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ and $\boldsymbol{p}_4$, one needs to check that the diagonals lie between the edges. For example, the vector $\boldsymbol{p}_4 - \boldsymbol{p}_2$ should be between the vectors $\boldsymbol{p}_1 - \boldsymbol{p}_2$ and $\boldsymbol{p}_3 - \boldsymbol{p}_2$, all



(a) Input of the program.

(b) Unstructured grid.

(c) Initial triangulation.

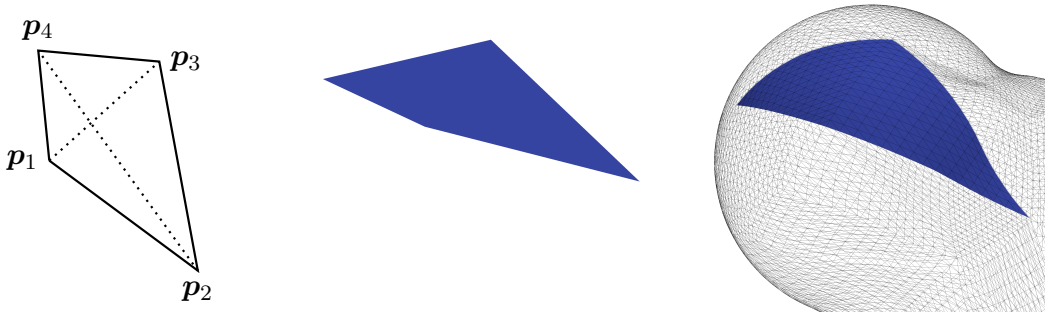
(d) Improved initial triangulation.

Figure 4.33. First steps of the mesh generation.

located in the origin, and likewise the vector $\boldsymbol{p}_1 - \boldsymbol{p}_3$ should be between the vectors $\boldsymbol{p}_2 - \boldsymbol{p}_3$ and $\boldsymbol{p}_4 - \boldsymbol{p}_3$, also located in the origin. Given three vectors in $\mathbb{R}^3$ located in the origin, $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, the test checks if $\boldsymbol{c}$ is between $\boldsymbol{a}$ and $\boldsymbol{b}$ by checking if the following conditions are fulfilled:

$$\begin{cases} \langle \boldsymbol{a} \times \boldsymbol{b}, \boldsymbol{a} \times \boldsymbol{c} \rangle \geq 0, \\ \langle \boldsymbol{b} \times \boldsymbol{c}, \boldsymbol{b} \times \boldsymbol{a} \rangle \geq 0. \end{cases}$$

The first equation checks whether the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ are on the same side of the vector $\boldsymbol{a}$. This is done mainly through the calculation of the cross product. If the sine changes sign, the vector from the cross product will be pointing in the opposite direction. The second equation, similarly, checks that the vectors $\boldsymbol{a}$ and $\boldsymbol{c}$ are on the same side of the vector $\boldsymbol{b}$. Since the surface is a three dimensional construct, the vertices of the polygon need to be projected onto a plane to check the position of the diagonals. The plane chosen is the median plane resulting from the arithmetic mean of the normals of the two triangles involved and the common points.



(a) Simple plane convexity for a given quadrangle.

(b) Representation of the quadrangle in the median plane.

(c) Representation of the quadrangle on the surface.

Figure 4.34. Convexity check by means of the diagonals.

Let the triangles be $\Delta_1 = \Delta(\tilde{\boldsymbol{p}_1}, \boldsymbol{p}_2, \boldsymbol{p}_4)$ and $\Delta_2 = \Delta(\boldsymbol{p}_2, \tilde{\boldsymbol{p}_3}, \boldsymbol{p}_4)$ with the common edge $(\boldsymbol{p}_2, \boldsymbol{p}_4)$, as depicted in fig. 4.35. Only two points need to be projected, namely the points that do not belong to the common edge, $\tilde{\boldsymbol{p}}_1$ and $\tilde{\boldsymbol{p}}_3$. Let the median plane be defined by $\boldsymbol{n}_{1/2} = (\boldsymbol{n}_{\Delta_1} + \boldsymbol{n}_{\Delta_2})/2$ and the point $\boldsymbol{p}_2$. It can be easily shown that the point $\boldsymbol{p}_4$ also belongs to the median plane. A point $\boldsymbol{x}$ belongs to the plane if it fulfils the plane equation

$$\langle \boldsymbol{n}_{1/2}, \boldsymbol{x} - \boldsymbol{p}_2 \rangle = 0.$$

We now insert the point, $\boldsymbol{p}_4$, into the plane equation and get the relation

$$\langle \boldsymbol{n}_{1/2}, \boldsymbol{p}_4 - \boldsymbol{p}_2 \rangle = \left\langle \frac{\boldsymbol{n}_{\Delta_1} + \boldsymbol{n}_{\Delta_2}}{2}, \boldsymbol{p}_4 - \boldsymbol{p}_2 \right\rangle = \frac{\langle \boldsymbol{n}_{\Delta}, \boldsymbol{p}_4 - \boldsymbol{p}_2 \rangle + \langle \boldsymbol{n}_{\Delta_2}, \boldsymbol{p}_4 - \boldsymbol{p}_2 \rangle}{2}.$$

Since the points, $\boldsymbol{p}_2$ and $\boldsymbol{p}_4$, belong to both planes defined by Δ_1 and Δ_2, they satisfy the plane equation for each of the triangles taken separately and thus also the plane

equation of the median plane

$$\langle \boldsymbol{n}_{1/2}, \boldsymbol{p}_4 - \boldsymbol{p}_2 \rangle = 0.$$

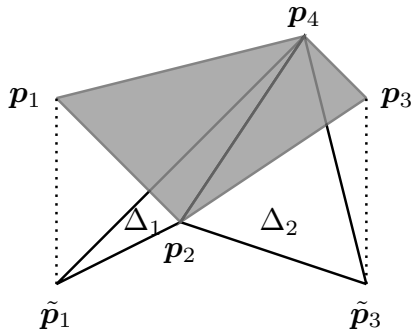


Figure 4.35. Projection for the plane convexity check for two triangles.

This naive approach works well when the triangles are in the same plane to begin with, but fails miserably otherwise. This is illustrated in fig. 4.34b and fig. 4.34c, where the projected triangle vertices form a convex quadrangle while the partial surface has an angle seemingly close to π.

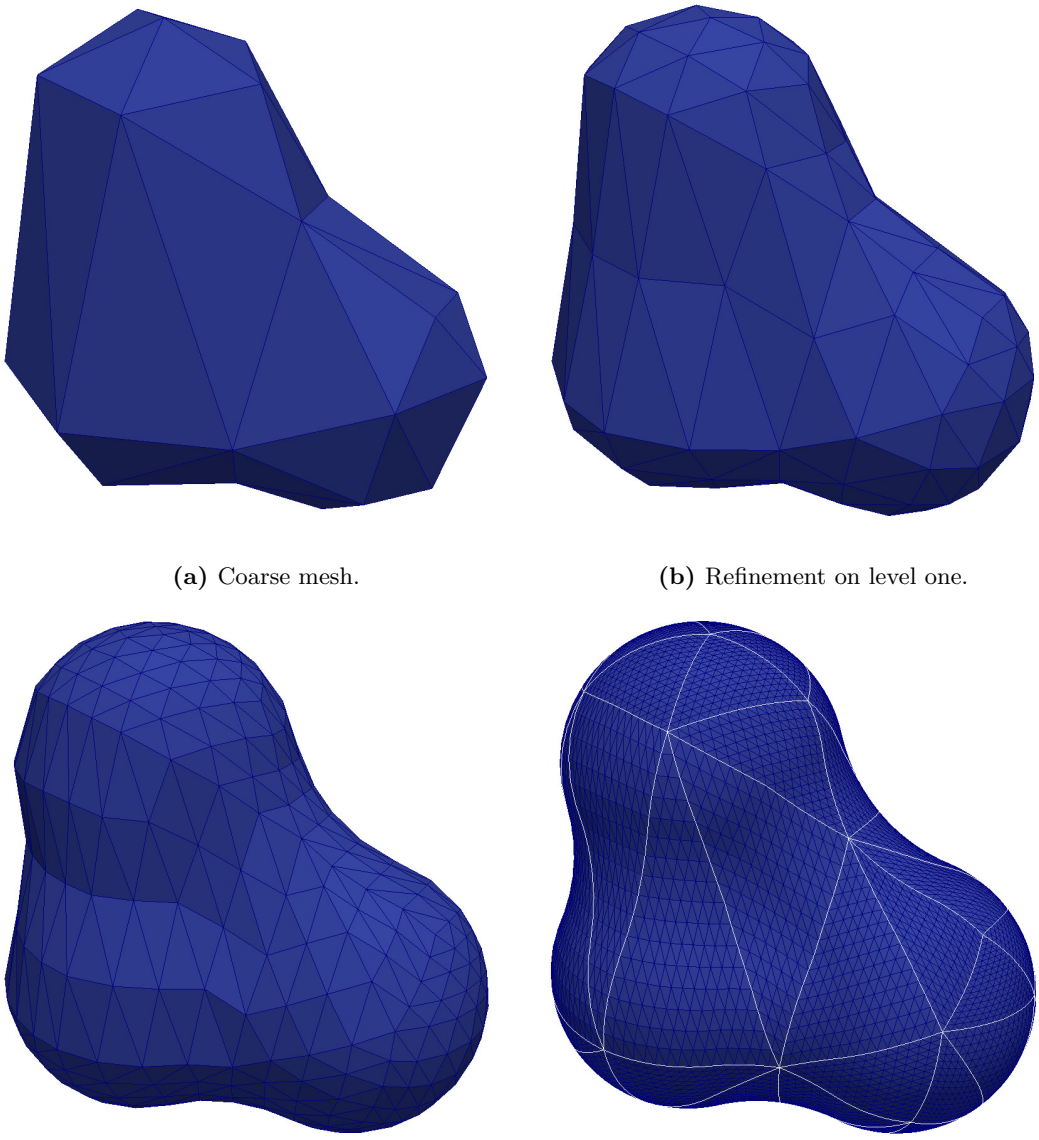
A different approach for checking convexity is to consider the angles between the edges of the projected polygon. This means that the angles should be as close to $\pi/2$ as possible. Similar results as the check for the diagonals can be achieved. Since this approach can fail depending on the median of the normals chosen, a different method needs to be used.

4.4.1.2. Convexity on the surface

The approach used in our implementation approximates the angles between the edges on the surface instead of a plane. Patches should be chosen such that the resulting elements form a nice mesh on the surface, i.e., a mesh without degenerated elements. This means that the angles on the surface should be as close to $\pi/2$ as possible.

In order to approximate the angles on the surface, points on the edges are generated by a uniform refinement of the coarse triangles as shown in fig. 4.32a up to a given level `patchOptimize`. Fig. 4.36 shows the successive uniform discretization of the coarse mesh into a mesh of level one, a mesh of level two, and the final mesh of level four. The new points generated by the splitting of the triangles are projected in the direction of the normal onto the surface. For each point, as in eq. (4.25), the mean of the normals of the two triangles to which the edge belongs to is being used.

Finding the order of the points on the edges can be naively done by using the Dijkstra algorithm, [16], once for each sought path. The nodes and edges of the graph are in this case the vertices and edges of the fine mesh. The start and end node for one run of the Dijkstra algorithm would then be the nodes representing the ends of the path. The



(a) Coarse mesh.

(b) Refinement on level one.

(c) Refinement on level two.

(d) Edges and fine mesh on level four.

Figure 4.36. Coarse mesh and refinement up to the required level for the optimization, `patchOptimize`.

algorithm finds the shortest path between the nodes with a complexity of up to $O(n_n^2)$ per path, where n_n is the number of nodes in the graph, being equal to the number of vertices in the fine mesh.

We use a more clever algorithm that stores an additional integer per node in the fine mesh where the corresponding index of the edge in the coarse mesh is stored, as depicted in fig. 4.37. In case the node is a node of the coarse mesh, the particular value of -1 is used. Special attention is given to the first refinement, which initializes the first nodes corresponding to the first vertices that are inserted on the edges. The new nodes are assigned the edge index of the coarse mesh edge that they belong to and are inserted in a path vector for the edge under consideration. Every subsequent refinement checks the value assigned to the vertices of the fine edge to be refined. For one edge with node indices, $e = (n_1, n_2)$, the midpoint generated by the refinement checks the values of the help vector at the indices n_1 and n_2, `helpEdgeVector`$[n_1]$ and `helpEdgeVector`$[n_2]$, respectively. If the values are equal, the new node belongs to the same coarse edge and the new node is marked accordingly and inserted into the path vector. The node also belongs to the coarse edge if one of the vertices is a vertex of the coarse mesh marked with -1. The new node is otherwise marked with -2 and is an inner node. This is done in constant time while calculating the refinements.

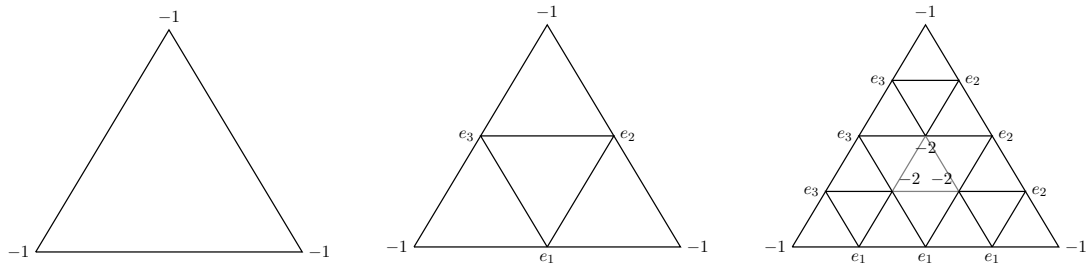


Figure 4.37. Refinement of the elements with path information.

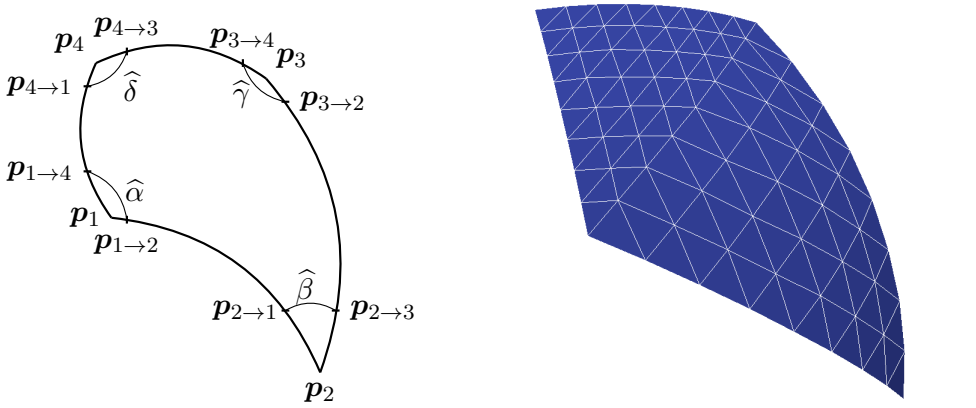
By repeating this procedure recursively, the points on the edges, highlighted in white in fig. 4.36d, can be determined. The runtime of this refinement step can be improved by refining the elements only towards the boundary. This is however not done here, since the tests showed that the refinement time is not the bottleneck of the mesh generation.

The angles are finally approximated by calculating them using points on the edges under consideration, as schematically shown in fig. 4.38a.

The convexity of the respective patch can, in this case, be checked by the following equations:

$$\begin{cases} \widehat{\alpha} \leq \pi : \langle (\boldsymbol{p}_{1\to 4} - \boldsymbol{p}_1) \times (\boldsymbol{p}_{1\to 2} - \boldsymbol{p}_1), \boldsymbol{n}_{\boldsymbol{p}_1} \rangle \geq 0, \\ \widehat{\beta} \leq \pi : \langle (\boldsymbol{p}_{2\to 1} - \boldsymbol{p}_2) \times (\boldsymbol{p}_{2\to 3} - \boldsymbol{p}_2), \boldsymbol{n}_{\boldsymbol{p}_2} \rangle \geq 0, \\ \widehat{\gamma} \leq \pi : \langle (\boldsymbol{p}_{3\to 2} - \boldsymbol{p}_3) \times (\boldsymbol{p}_{3\to 4} - \boldsymbol{p}_3), \boldsymbol{n}_{\boldsymbol{p}_3} \rangle \geq 0, \\ \widehat{\delta} \leq \pi : \langle (\boldsymbol{p}_{4\to 3} - \boldsymbol{p}_4) \times (\boldsymbol{p}_{4\to 1} - \boldsymbol{p}_4), \boldsymbol{n}_{\boldsymbol{p}_4} \rangle \geq 0. \end{cases} \tag{4.39}$$

Here $\boldsymbol{n}_{\boldsymbol{p}_i}$ is the normal at point $\boldsymbol{p}_i$ calculated as the mean of the normal of all elements



(a) Convexity check using surface angles. **(b)** Convexity check in three dimensions.

Figure 4.38. Convex check by means of angles.

on the fine mesh containing $\boldsymbol{p}_i$:

$$\boldsymbol{n}_{\boldsymbol{p}_i} = \frac{\sum_{\boldsymbol{p}_i \in \Delta} \boldsymbol{n}_\Delta}{\deg \boldsymbol{p}_i}.$$

4.4.2. Combining triangles to quadrangles

Even if all non-convex combinations are left out, each triangle has up to three possible combinations for generating quadrangular patches. The challenge is to find out which triangles should be merged to a patch such that the overall mesh fitness is the best. The dual graph can be used to transform the problem into a graph based problem. The dual graph is defined as the graph where each node represents a triangle in the mesh. The neighbouring relations in the mesh form the edges of the graph. The edges can be equipped with weights corresponding to the quality of the patch generated by the two underlying triangles. A perfect matching represents a choice of edges in the graph such that all nodes are covered by exactly one edge in the matching. A matching is the equivalent of a valid choice of patch generation. The best matching is sought after such that the overall fitness is minimized. To this end, the non-convex combinations are penalized by the maximum possible weight of 10^6. For the convex combinations, the weight is set by the fitness measure chosen. Several fitness functions have been investigated. We will present them here shortly and analyse their advantages.

4.4.2.1. Plane fitness measure

The first type of fitness measure looks at the projected vertices of the patch under consideration. Let the triangles for which the fitness measure is calculated be defined

by $\Delta = \Delta(\tilde{\boldsymbol{p}_1}, \boldsymbol{p}_2, \boldsymbol{p}_4)$ and $\Delta_2 = \Delta(\boldsymbol{p}_2, \tilde{\boldsymbol{p}_3}, \boldsymbol{p}_4)$ with the common edge $(\boldsymbol{p}_2, \boldsymbol{p}_4)$, as depicted in fig. 4.35.

The plane onto which the triangles are projected is determined by the mean of the normals

$$\boldsymbol{n}_{1/2} = \frac{\boldsymbol{n}_{\Delta_1} + \boldsymbol{n}_{\Delta_2}}{2}$$

and the two common points $\boldsymbol{p}_2$ and $\boldsymbol{p}_4$. The only points that need to be projected onto this plane are the two points $\tilde{\boldsymbol{p}_1}$ and $\tilde{\boldsymbol{p}_3}$. These are projected onto $\boldsymbol{p}_1$ and $\boldsymbol{p}_3$, respectively.

The first of the *plane* fitness measures takes the area, $A(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4)$, and the perimeter, $P(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4)$, of the quadrangle into consideration:

$$fit(\Delta_1, \Delta_2) = \frac{P(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4)^2}{16A(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4)}.$$

This function is one for a square and has values between zero and one for all other quadrangles.

An alternative version of the *plane* fitness measure is to consider the angles between the vertices of the patch. Recall that the sine has values in $[0, 1]$ for angles between 0 and π with a maximum at $\pi/2$. The square of the sine can be calculated by means of the dot product of adjacent edges. This leads to the fitness measure given by the algorithm found in lst. 4.40. It calculates the inverse of the square of the sine of the angles and penalizes the patches with angles further away from $\pi/2$.

Listing 4.40 *Plane* fitness by means of angles.

```
Eigen::Vector3d a = (p1-p2).normalized();
Eigen::Vector3d b = (p3-p2).normalized();
Eigen::Vector3d c = (p3-p4).normalized();
Eigen::Vector3d d = (p1-p4).normalized();

double sin_alpha = 1./(1-a.dot(d)*a.dot(d));
double sin_beta  = 1./(1-a.dot(b)*a.dot(b));
double sin_gamma = 1./(1-b.dot(c)*b.dot(c));
double sin_delta = 1./(1-c.dot(d)*c.dot(d));

double max;
if(sin_alpha < sin_beta) max = sin_beta;
if(max < sin_gamma) max = sin_gamma;
if(max < sin_delta) max = sin_delta;
return fabs(max);
```

4.4.2.2. Normal fitness measure

The category of the *plane* fitness measures have the disadvantage of not taking the curvature of the surface into account. This leads in general to a bad combination of patches, which might look good in the median plane, where the fitness has been calculated, but that are distorted into ugly combinations on the surface, as seen previously for the convexity check in fig. 4.34. By using the normals of the coarse triangles, one can favour pairs of triangles that have a small difference in the normals. This can be expressed through the fitness measure

$$fit_{\boldsymbol{n}}(\Delta_1, \Delta_2) = \frac{1}{1 - \langle \boldsymbol{n}_{\Delta_1}, \boldsymbol{n}_{\Delta_2} \rangle^2}.$$

This means that triangles which are almost in the same plane are preferred over triangles with a big difference in the angle between the normals, which would in turn lead to a resulting patch with a high curvature. This however does not take the shape of the triangles and the resulting quadrangle into account.

4.4.2.3. Surface fitness measure

The third type of fitness measure presented is a *surface* fitness measure, where the surface angles of the patches are taken into account. This leads to the best results and is used in the code. Such a fitness measure can be constructed by using the same points as for the convexity check seen fig. 4.38a. This enables us to approximate the sine of the surface angles in accordance with

$$\begin{cases} \sin^2 \widehat{\alpha} \approx 1 - \left(\dfrac{\langle (\boldsymbol{p}_{1\to4} - \boldsymbol{p}_1), (\boldsymbol{p}_{1\to2} - \boldsymbol{p}_1) \rangle}{\|\boldsymbol{p}_{1\to4} - \boldsymbol{p}_1)\|_2 \cdot \|\boldsymbol{p}_{1\to2} - \boldsymbol{p}_1\|_2} \right)^2, \\ \sin^2 \widehat{\beta} \approx 1 - \left(\dfrac{\langle (\boldsymbol{p}_{2\to1} - \boldsymbol{p}_2), (\boldsymbol{p}_{2\to3} - \boldsymbol{p}_2) \rangle}{\|\boldsymbol{p}_{2\to1} - \boldsymbol{p}_2\|_2 \cdot \|\boldsymbol{p}_{2\to3} - \boldsymbol{p}_2\|_2} \right)^2, \\ \sin^2 \widehat{\gamma} \approx 1 - \left(\dfrac{\langle (\boldsymbol{p}_{3\to2} - \boldsymbol{p}_3), (\boldsymbol{p}_{3\to4} - \boldsymbol{p}_3) \rangle}{\|\boldsymbol{p}_{3\to2} - \boldsymbol{p}_3\|_2 \cdot \|\boldsymbol{p}_{3\to4} - \boldsymbol{p}_3\|_2} \right)^2, \\ \sin^2 \widehat{\delta} \approx 1 - \left(\dfrac{\langle (\boldsymbol{p}_{4\to3} - \boldsymbol{p}_4), (\boldsymbol{p}_{4\to1} - \boldsymbol{p}_4) \rangle}{\|\boldsymbol{p}_{4\to3} - \boldsymbol{p}_4\|_2 \cdot \|\boldsymbol{p}_{4\to1} - \boldsymbol{p}_4\|_2} \right)^2. \end{cases}$$

The angles are closest to $\pi/2$ if

$$fit_{\square} = \frac{1}{\min\{\sin^2 \widehat{\alpha}, \sin^2 \widehat{\beta}, \sin^2 \widehat{\gamma}, \sin^2 \widehat{\delta}\}}, \tag{4.41}$$

is closest to one. A similar algorithm as in lst. 4.40 is exploited for calculating the sine of the angles.

4.4.2.4. The blossom algorithm

The blossom algorithm, [41], is used to find a perfect matching. Recall that a matching is a set of disjoint edges and a perfect matching is a matching where all the nodes are

being covered. In particular, this means that each triangle is matched to one of its neighbours. In our case, there are several perfect matchings possible. The one which optimizes the cost functional, represented here by the fitness measure, is being chosen.

The blossom algorithm finds the perfect matching, if one such matching exists, in a complexity $O(nm \log n)$, where n is the number of nodes in the graph and m is the number of edges. For the case of the triangular mesh, the number of nodes in the graph is equal to the number of triangles in the mesh, $n = n_\Delta$. A triangle has three neighbours, which leads to a total number of neighbours, for a closed surface, given by $3n_\Delta/2$. The number of edges in the graph used for the matching is thus given by the number of neighbouring relations, $m = 3n_\Delta/2$. This leads to a complexity of $O(n_\Delta^2 \log n_\Delta)$, with $n_\Delta = 2n_\square$ being in general a relatively small number.

4.5. Surface parametrization

Another input parameter sets the number of desired refinements, n_L, of the coarse surface representation. It defines the number of levels in the resulting quadrangular mesh. The starting point of the patch parametrization is the coarse mesh depicted in fig. 4.36a together with the output of the blossom algorithm. The refinement is done iteratively until the mapping from the unit square to the patch described in chpt. 3, in particular in fig. 3.1, can be calculated by interpolation. This yields a subdivision scheme which is illustrated in fig. 4.43. For the subdivision scheme to work, we define an element as the smallest bilinear quadrangular unit on the current level of refinement. In particular, this means that the coarsest level has $n_\square$ elements. Each refinement step divides each element of the surface mesh into four new elements.

In order to achieve the refinement on level n_L, one recursively refines the elements. This is done by using the mapping from the unit square, $\square$, to the bilinear element, described by its points $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_3$, and $\boldsymbol{p}_4$. The element is refined and the new points are lifted onto the surface in normal direction. The bilinear interpolation between the points $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_3$, and $\boldsymbol{p}_4$ is given by

$$\boldsymbol{Q}_{\square_i}(s,t) = (1-s)((1-t)\boldsymbol{p}_1 + t\boldsymbol{p}_4) + s((1-t)\boldsymbol{p}_2 + t\boldsymbol{p}_3). \tag{4.42}$$

The surface point corresponding to $(s,t) \in \square$ is calculated by projecting the result of the bilinear interpolation in the direction of the normal at this point. The normal is calculated by taking the partial derivatives in s and t and applying the cross product

$$\boldsymbol{n}_{\square_i}(s,t) = \frac{\partial \boldsymbol{Q}_i}{\partial s}(s,t) \times \frac{\partial \boldsymbol{Q}_i}{\partial t}(s,t).$$

On the boundary between two elements the arithmetic mean between the normals from both sides is taken.

The first image of fig. 4.43 illustrates the coarse mesh including the normals at the patch corners and the element normal calculated in the centre of the element. The other images show the next refinements. The points are calculated by taking the bilinear interpolation from (4.42) and evaluating it in the element vertices. In particular, for

one refinement step, the inner point found at $(1/2, 1/2)$ and the boundary points at $(0, 1/2)$, $(1, 1/2)$, $(1/2, 0)$, and $(1/2, 1)$ have to be computed for each element. All points on the element boundaries are calculated only once.

4.6. Mesh improvement

If the initial position of the vertices on the coarse quadrangular representation of the surface does not lead to satisfying results, improvements of the mesh can be applied by repositioning the vertices of the coarse mesh. These improvements do not allow for changes in the underlying matching, but move the vertices instead such that the fitness of the patches is improved. The improvement step is done on the `patchOptimize` level of refinement.

4.6.1. Doubly neighbours

The first improvement treats the case of patches with two common edges, seen in fig. 4.44. This has to be avoided and treated before any improvement algorithm can run. It cannot be optimized by discrete movements of the vertices of the coarse mesh, since in each step the point that belongs to both adjacent edges will be moved to optimize one patch while worsening the fitness of the other patch. We overcome this situation by removing the common point in the coarse representation altogether and using less patches to discretize the surface. This step is performed on the output of the blossom algorithm before the mesh improvement starts.

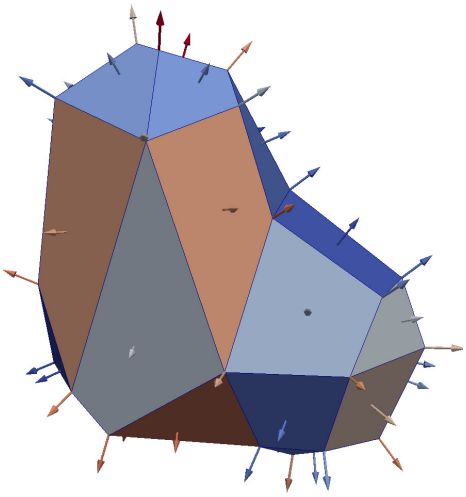
4.6.2. Angle improvement

The best discretization has the same surface angle for each angle attached to a vertex. In order to achieve that, the vertex nodes of the coarse mesh are being moved, such that the difference in the angles is decreased as seen schematically in fig. 4.48. In the continuous approach, the point $\boldsymbol{p}_i$ would be replaced by the point $\tilde{\boldsymbol{p}}_i$ which minimizes $\sum_j (\alpha_j - \alpha_{opt})^2$, where α_j are the surface angles measured as in fig. 4.38a. The optimal angle, α_{opt}, depends on the degree, $\deg \boldsymbol{p}_i$, of the coarse node, $\boldsymbol{p}_i$, with respect to the quadrangular patches which contain the node: $\alpha_{opt} = {}^{2\pi}/_{\deg \boldsymbol{p}_i}$. Two cases arise, the convex and the non-convex surface angle case.

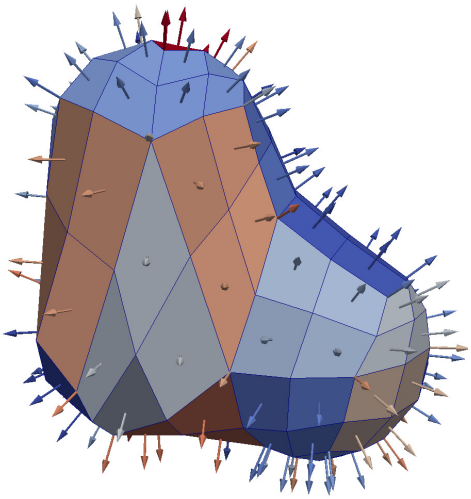
4.6.2.1. Convex improvement

If all the angles belonging to a node are smaller than π, the functional containing the squared difference in the cosine value is used. This can easily be calculated by taking the dot product of the normalized edges that span the angle under consideration:

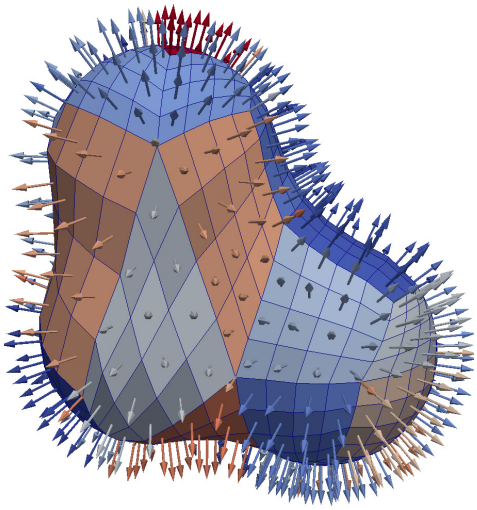
$$fit(\boldsymbol{p}_i) = \frac{\sum_j (\langle \boldsymbol{p}^j_{i\to i-1} - \boldsymbol{p}_i, \boldsymbol{p}^j_{i\to i+1} - \boldsymbol{p}_i \rangle - \cos(\alpha_{opt}))^2}{\deg \boldsymbol{p}_i}. \tag{4.46}$$



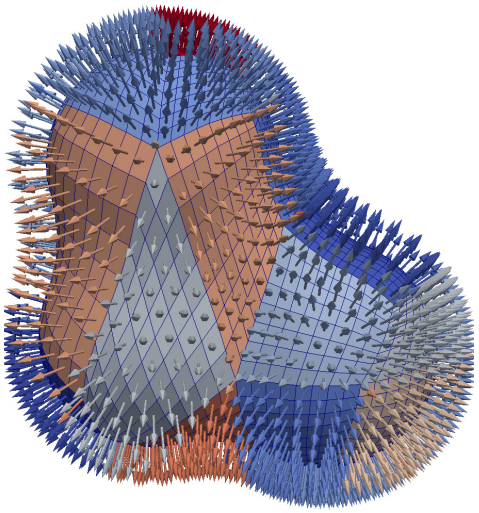
(a) Coarse mesh, including element normals.



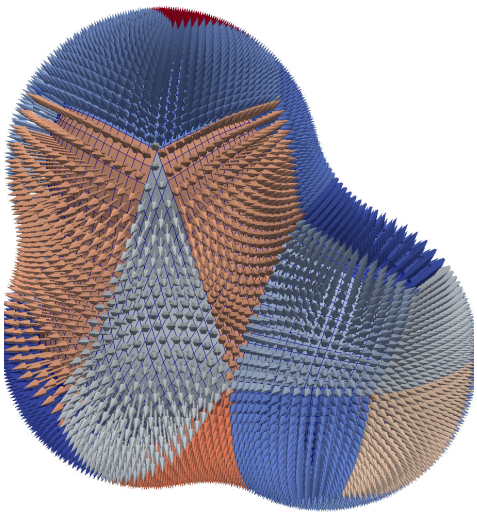
(b) Refinement on level one, including element normals.



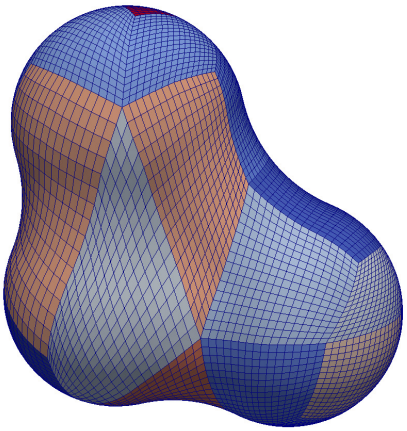
(c) Refinement on level two, including element normals.



(d) Refinement on level three, including element normals.



(e) Refinement on level four, including element normals.



(f) Refinement on level four.

Figure 4.43. Coarse patches and refinement up to required level.

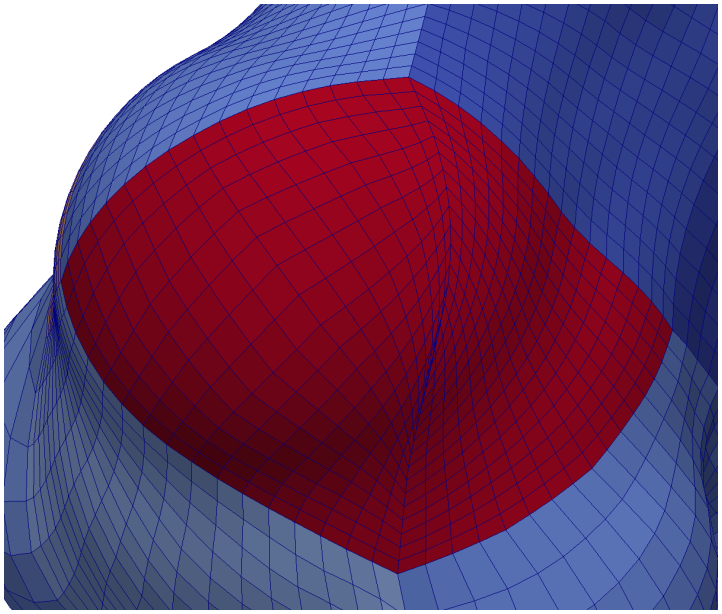


Figure 4.44. Problem with local optimization in case of two patches with two common edges.

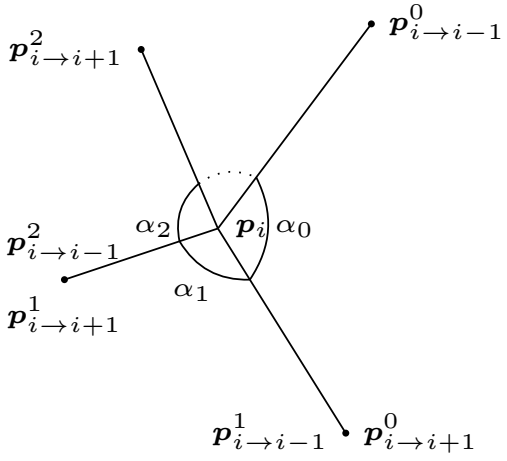


Figure 4.45. Description of angles around a patch vertex.

Here, each angle of the node, $\boldsymbol{p}_i$, taken inside the patch j is schematically seen in fig. 4.45 and thus approximated by $\measuredangle(\boldsymbol{p}^j_{i\to i-1}, \boldsymbol{p}_i, \boldsymbol{p}^j_{i\to i+1})$ on the finest level. The direction, in which the point $\boldsymbol{p}_i$ is moved, is the direction of steepest descent given by

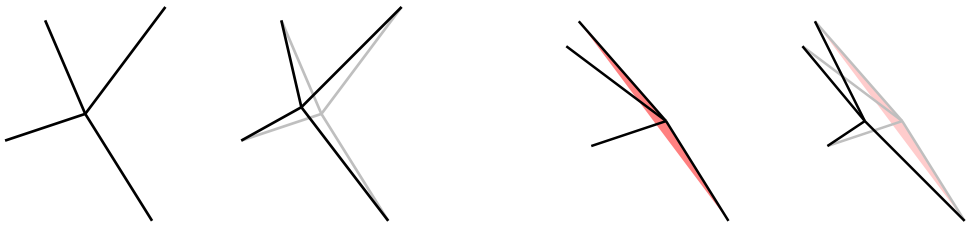
$$\nabla fit(\boldsymbol{p}_i) = \frac{2\sum_j(\langle \boldsymbol{p}^j_{i\to i-1} - \boldsymbol{p}_i, \boldsymbol{p}^j_{i\to i+1} - \boldsymbol{p}_i\rangle - \cos(\alpha_{opt}))(2\boldsymbol{p}_i - \boldsymbol{p}^j_{i\to i-1} - \boldsymbol{p}^j_{i\to i+1})}{\deg \boldsymbol{p}_i}. \quad (4.47)$$

The step size, h, used for the update depends on the length of the shortest edge, e, between two vertices belonging to the coarse mesh, taken with respect to the polygonal approximation of the surface given by the fine quadrangular mesh, $h = \min_e \|e\|/2$. If the new point $\tilde{\boldsymbol{p}}_i = \boldsymbol{p}_i - h\nabla fit(\boldsymbol{p}_i)$ is outside of the geometry, the bisection algorithm from algo. 4.16 can be used to find the closest point on the surface in the direction of steepest descent. If the point $\tilde{\boldsymbol{p}}_i$ is inside the geometry, the normal direction at the point $\boldsymbol{p}_i$, $\boldsymbol{n}_{\boldsymbol{p}_i}$, is used to lift the point onto the surface.

If $\|\nabla fit(\boldsymbol{p}_i) - \langle \nabla fit(\boldsymbol{p}_i), n_{\boldsymbol{p}_i}\rangle n_{\boldsymbol{p}_i}\|_2 \approx 0$, the update would require a movement exclusively in the normal direction of $\boldsymbol{p}_i$ and the update is not performed.

4.6.2.2. Non-convex improvement

The situation changes if one of the angles is not convex. Convexity can be checked as in eq. (4.39). In this case, the value of the other angles does not play such an important role and the sole purpose of the improvement is to reduce the non-convex angle. This can be done by moving the point in the direction of the bisector of the angle that needs to be improved.

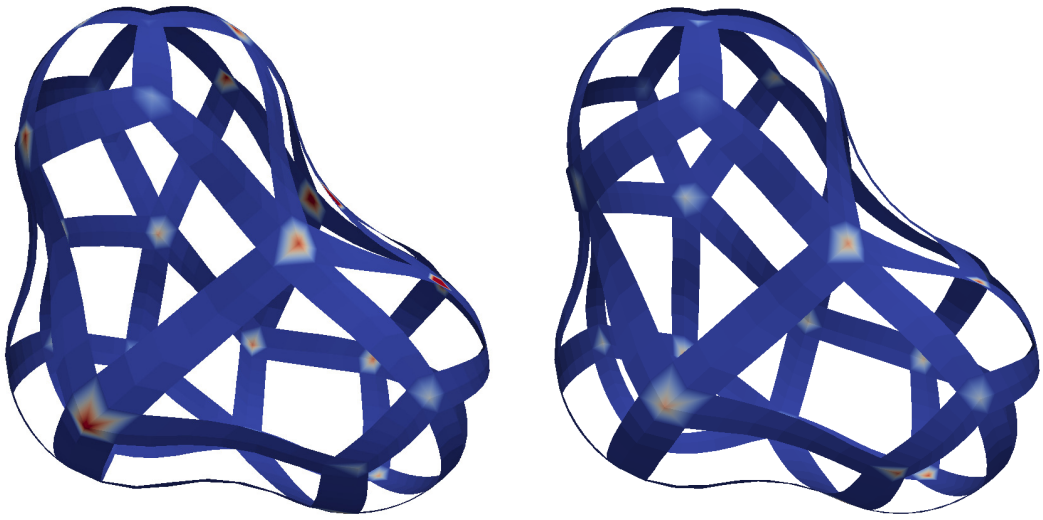


(a) Moving a convex surface angle according to eq. (4.47).

(b) Moving a non-convex surface angle in the direction of bisector of large angle.

Figure 4.48. Moving points to improve surface angles.

The improvements to the angles are summarized schematically in fig. 4.48. The left subfigure shows the convex case, where the direction is given by eq. (4.47), while the right subfigure shows the case of one non-convex angle. Please note that the program does not always perform as expected in the case of non-convex angles, since the projection to the surface of the molecule leads often to a self-intersecting mesh. The program still tries to improve the situation, the user is however notified and the result should be checked thoroughly.



(a) Initial boundary mesh with worst node fitness 0.7098.

(b) Final boundary mesh with worst node fitness 0.2505.

Figure 4.49. Initial and final boundary mesh after mesh improvement for a toy example.

4.6.2.3. Extreme obtuse or acute angle

The fitness measure described in eq. (4.46) fails in one situation. Let one node have the degree three, one of the angles be close to π, and the other two angles close to $\pi/2$. A simple calculation leads to the approximate value of the fitness function

$$fit(\boldsymbol{p}_i) = \frac{\frac{1}{4} + \frac{1}{4} + (-1 + \frac{1}{2})^2}{3} = \frac{1}{4} = 0.25.$$

This is why, in the case of angles with extreme values for the cosine, for example $|\cos\alpha| > 0.8$, a similar approach as for the non-convex angles is taken, namely, the update direction is set to be equal to the direction of the bisector, if the cosine is positive, or the opposite direction of the bisector, if the cosine is negative. This direction would then optimize only the angle under consideration. the threshold of 0.8 can be changed in the file containing all the relevant parameters.

4.6.3. Null length edge

The direction of steepest descent as described in eq. (4.47) does not take into account the lengths of the edges in the mesh. This is in general not a big problem, but, when an edge is in a configuration similar to that seen in fig. 4.50, the resulting steepest descent direction would push the points closer and closer together, which can result in a self-intersecting mesh. The projected gradient scheme can be used in this case by a restricting the contraction of the edges. When this option is enabled, the lengths of the edges on the surface on the level for the optimization, `patchOptimize`, are calculated. If a node is connected to an edge that is very small, relative to the largest edge in the

mesh calculated on the first configuration, the direction of this edge is removed from the steepest descent direction if this would lead to an even smaller edge. This can be done by calculating the dot product between the normalized edge direction $\boldsymbol{e}_j$ and the direction that the point is moved in, $\nabla fit(\boldsymbol{p}_i)$. If the dot product is larger than zero, the direction of the edge is subtracted from the steepest descent direction:

$$\nabla \widehat{fit}(\boldsymbol{p}_i) := \nabla fit(\boldsymbol{p}_i) - \langle \nabla fit(\boldsymbol{p}_i), \boldsymbol{e}_j \rangle \boldsymbol{e}_j.$$

If the resulting direction after subtracting the normal component is almost zero, the update for the current node is aborted.

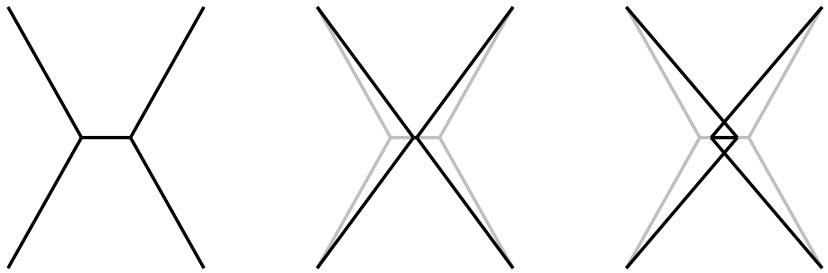


Figure 4.50. Situation where the improvement results in edges of length zero.

4.6.4. Improvement strategy

The improvement is done in two steps. In the first step, all nodes are moved which have an associated fitness larger than $1/2$ `stopNodeMoveglobal`, where

$$\texttt{stopNodeMoveglobal} = 0.2,$$

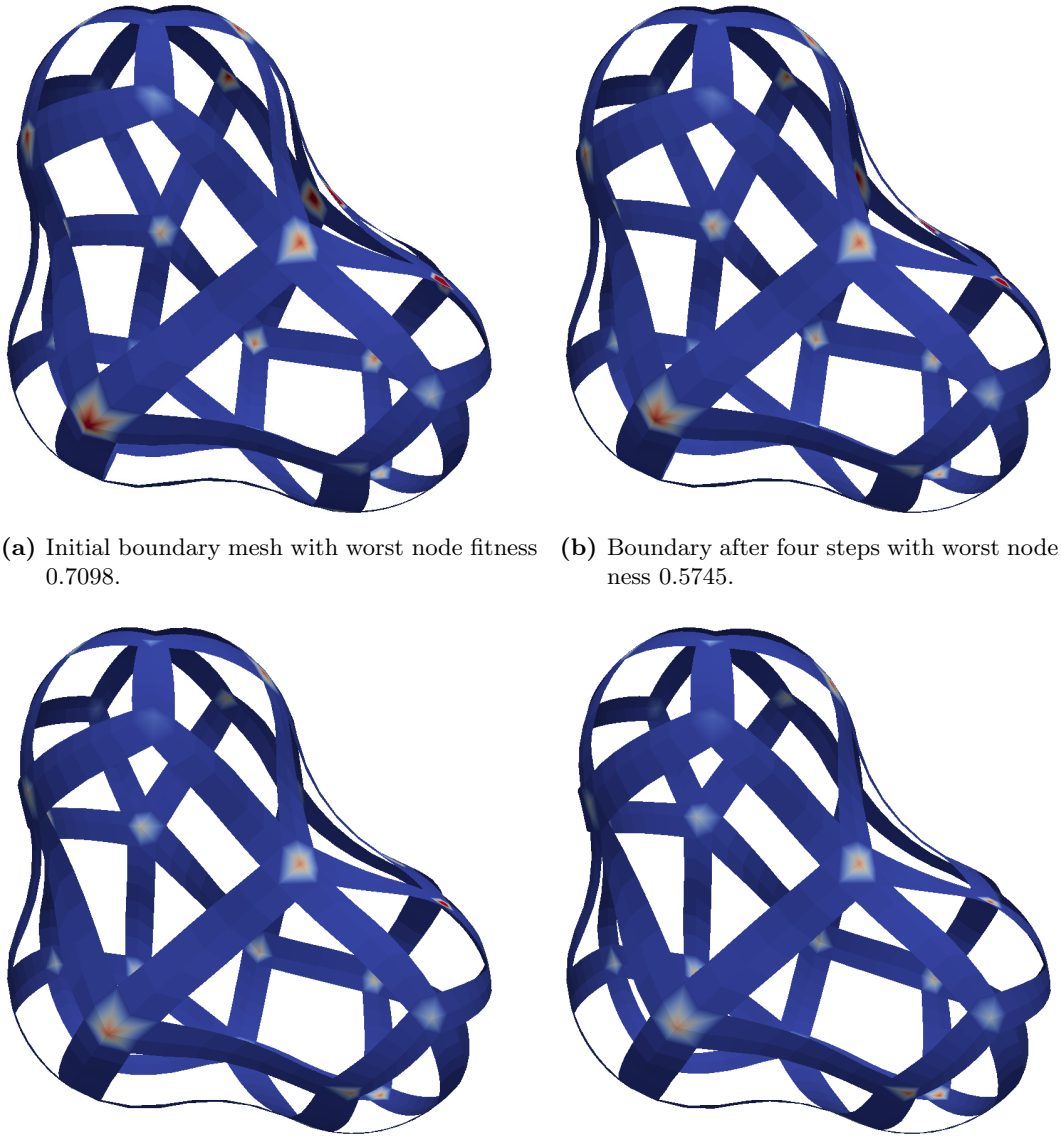
until an overall fitness smaller than `stopNodeMoveglobal` is reached. This step allows for some worsening iterations, where the new fitness is larger than the original one. If more than three worsening iterations are performed, the version of the mesh with the smallest fitness is kept.

The second improvement step moves only the worst node. This can change from iteration to iteration. The second step stops whenever the fitness is increased. After each movement, the relevant boundary points and elements are recomputed.

The global improvement can be seen in fig. 4.51, where the evolution of the mesh is seen for the toy example. A total of 12 global improvements steps have been performed and no local improvement was necessary. The fitness of the meshes is shown under each figure.

4.6.5. Runtime improvement

In each improvement step, only the points on the patch boundaries are considered. In order to improve the runtime, the mesh points are calculated in an adaptive manner



(a) Initial boundary mesh with worst node fitness 0.7098.

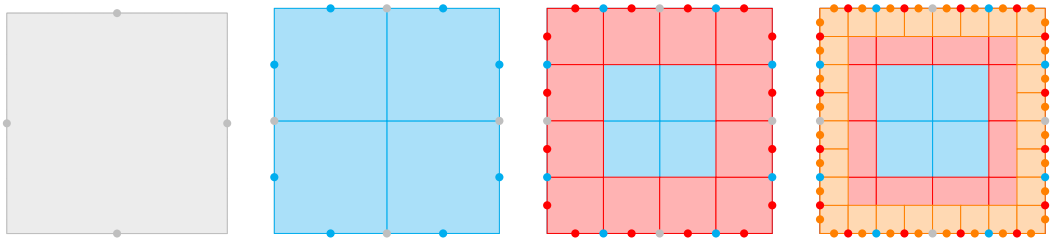
(b) Boundary after four steps with worst node fitness 0.5745.

(c) Boundary after eight steps with worst node fitness 0.4036.

(d) Final boundary mesh after twelve steps with worst node fitness 0.2505.

Figure 4.51. Improvement steps for the toy example.

only towards the boundary. The associated refinement can be seen in fig. 4.52. The information needed in order to compute the normals and thus the points on the boundary are the innermost elements on the next coarsest level than the level required for the boundary, `patchOptimize`. The complete refinement up to the levels requested is then done after the improvement step.



(a) Coarse mesh, level zero, and level one refinement of the boundary.

(b) Level one refinement of the inner boundary elements, level two refinement of the boundary.

(c) Level two refinement of the inner boundary elements, level three refinement of the boundary.

(d) Level three refinement of the inner boundary elements, level four refinement of the boundary.

Figure 4.52. Adaptive refinement for quadrangles.

4.7. Summary

The control flow diagram found in fig. 4.53 summarizes the order in which the operations are being performed. The first column includes the reading of the input file and the initialization of the geometry. The geometrical constructs needed by the evaluation of the level set function are computed. This includes the circles, the circle points, the tori, the description of the spherical concave patches and the domain decomposition.

The second column contains the generation of the coarse mesh. The marching cubes algorithm is first used to generate an approximation of the surface. The resulting unstructured grid is divided into triangles and then improved by applying the coarsening through edge replacement as found in algo. 4.22. The measure for the fitness of the triangles is given in eq. (4.18). For generating quadrangular patches we ensure an even number of triangles in the mesh by refining each triangle into four triangles, as seen in fig. 4.32a. Since $n_\square$ patches are requested, the triangular mesh is coarsened further until $n_\triangle = 2n_\square$ is reached. If the automatic patch detection is enabled, the number of patches is determined by the highest fitness of the mesh between the refinement into an even number of triangles and the coarse mesh.

The green column of the mesh generation control flow diagram shows the steps needed from the coarse mesh to the final output. Firstly, the coarse mesh is uniformly refined in order to use the surface fitness function found in eq. (4.41) and the convexity check from eq. (4.39). The blossom algorithm is then used to calculate a minimum cost matching which determines the triangles that are paired to a patch. Finally, the resulting quadrangular mesh is improved with respect to the fitness found in eq. (4.46), by taking all the special cases into account: the doubly neighbours, the the non-convex

angles, the extreme obtuse or acute anclgs, the null length edge. This is all done in each iteration of the improvement strategy applied.

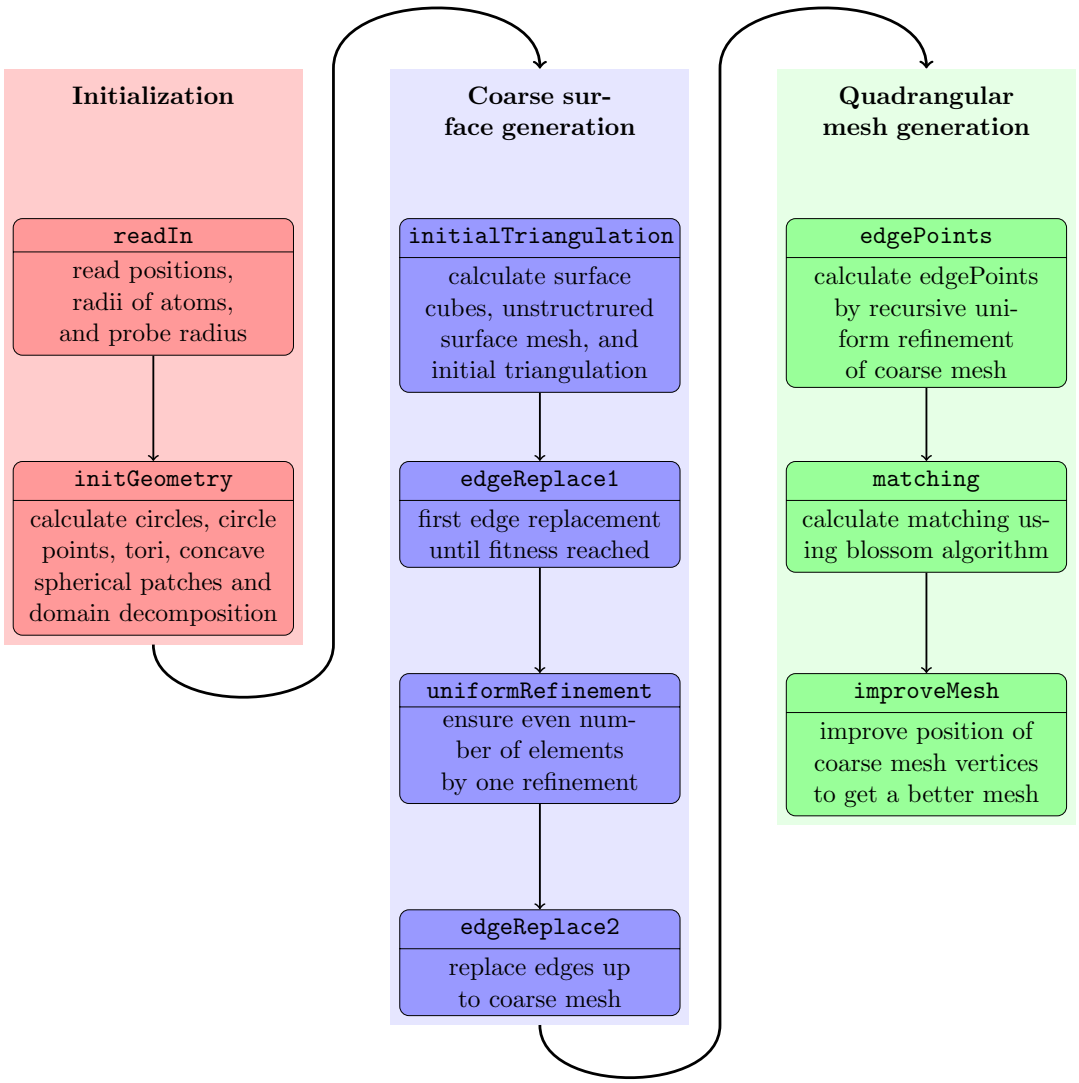


Figure 4.53. Control flow for the cavity generator.

CHAPTER 5

Validation

This chapter is concerned with the validation of the mesh generator on one hand and of the wavelet BEM solver on the other hand. The validation of the wavelet BEM solver is done by computing different levels for a small sample of molecules, solving the PCM equation found in eq. (1.9) on this mesh and comparing the result to the exact analytical value as obtained by the Gauss' theorem,

$$\Delta\sigma = \sigma_{\text{calc}} - \sigma_{\text{exact}}, \qquad \sigma_{\text{exact}} = \frac{1-\varepsilon}{\varepsilon}\sum_i Q_i, \tag{5.1}$$

where ε is the relative permittivity and Q_i are the nuclear point charges localized in the centres of the atoms. This gives rise to the nuclear interaction energy. The validation of the mesh generator will asses the performance and the limits of the cavity generator described in chpt. 4. Since the mesh generator can be used for any geometries where a level set function or a characteristic function of the molecule is known, this chapter will first explore both, the geometry description given by atoms and radii, as tediously described in the previous chapter, and the geometry description by isosurfaces. The last part of this chapter touches also the application of the wavelet BEM to ionic solutions and liquid crystals.

5.1. Geometry description by atoms

The geometry description given by atoms as seen in chpt. 4 is tested by the generation of cavities for different molecules. The test molecules chosen in this work are ethyl alcohol, aspirine, benzylpenicillin, caffeine, cubane and glucose to cover a broad spectrum of different molecular geometries. The graphical output of the mesh generator will be shown here only for ethyl alcohol. For the other molecules, please refer to appendix A.

The input file for ethyl alcohol is shown in lst. 5.2. The first line contains the number of atoms in the molecule, followed by one line per atom containing the coordinates of the atom and the van der Waals radius. All the other input files can be found in appendix A.

Listing 5.2 Input file ethyl alcohol.

```
9
-3.89983        -0.51196        -0.05537        1.70
-2.37851        -0.46398         0.00593        1.70
-4.32950        -0.26380         0.93819        1.20
-4.23092        -1.53126        -0.34574        1.20
-4.27313         0.21628        -0.80625        1.20
-1.99314        -1.21202         0.73553        1.20
-1.94286         0.82372         0.34649        1.52
-1.96225        -0.72745        -0.98924        1.20
-2.06475         0.91513         1.32761        1.20
```

5.1.1. Parameter file

Lst. 5.3 summarizes the general input parameters used for all mesh generation calculations within this chapter. The points on the surface are calculated with a precision `tol_point`. At the end, after all optimizations are done, the final points are projected onto the surface with a higher precision `tol_end`. The values `improveValRawFine` and `improveValEvenFine` correspond to fitness values of triangles as calculated by eq. (4.18). The first value influences the number of coarsening iterations done before the triangles are refined into four triangles to ensure an even number of triangles in the mesh. The second value defines the point in time at which the recording of the fitness of the mesh starts. The coarsening is carried out until the number of triangles is equal to twice the required number of patches, `nPatches`. In the case of the automatic patch detection, the mesh with the best fitness is hereafter used as a coarse mesh and the number of patches is adapted. The level of refinement, `patchOptimize`, is used to calculate which triangles are merged in the patch generation step and defines also the level at which the mesh improvement is carried out. For the mesh improvement steps, the `globalOpt` flag is set, which means that the vertex improvement is first done for all vertices until the worst vertex fitness is smaller than `stopNodeMoveGlobal` or the number of iterations that worsen the fitness, `worseningIterationsGlobal`, is reached.

The flag `useEdgeLengthInfo` determines whether or not the contraction of an edge is allowed beyond `edgeLengthContraction` of the shortest edge in the initial mesh.

Listing 5.3 List of parameters for the cavity generator.

```
// scaling factor for radius of atoms
const double scalingFactorRadius = 1.2;

// tolerance for point comparison, or length
const double tol_point = 1e-4;
const double tol_end = 1e-7;

const double improveValRawFine = 0.5;
const double improveValEvenFine = 0.5;

// level for mesh improvement and patch generation
const int patchOptimize = 4;

// lambda for stopping Global Improvements
const bool globalOpt = true;
const double stopNodeMoveGlobal = 0.2;
const int worseningIterationsGlobal = 3;

const bool useEdgeLengthInfo = true;
const double edgeLengthContraction = 0.7;

// constant used to declare angles as dangerous
const double tol_cosine = 0.9; // cos(25) = 0.9063
```

5.1.2. Fitness versus number of patches

The first validation test shows the influence of the number of patches on the resulting mesh. The number of patches has been varied from 40 to 60. The quality of the mesh is taken as the worst value of the node fitness in the mesh as given in eq. (4.46). The parameters used for calculating the mesh are summarized in lst. 5.3. Additionally, the factor from eq. (4.14) is chosen to be `geometryFactor = 5` or `geometryFactor = 10` as mentioned in the legend. The summary can be found in fig. 5.5. The results from the automatic detection of patches are shown in fig. 5.4 for ethyl alcohol. See figures A.2, A.5, A.8, A.11, A.14, and A.17 in the appendix for the resulting parametrizations of the other molecules. Fig. 5.5 shows the final fitness calculated as in eq. (4.46) over the fixed requested number of patches. It can be seen that the fitness varies between 0.28 and 3.5 and does not directly correlate with the number of patches. This is due to the fact that the coarsening is governed by the error of the edge replacement and the resulting quadrangular patches are not taken into account.

The time needed to compute the surface is split into nine main components. The first

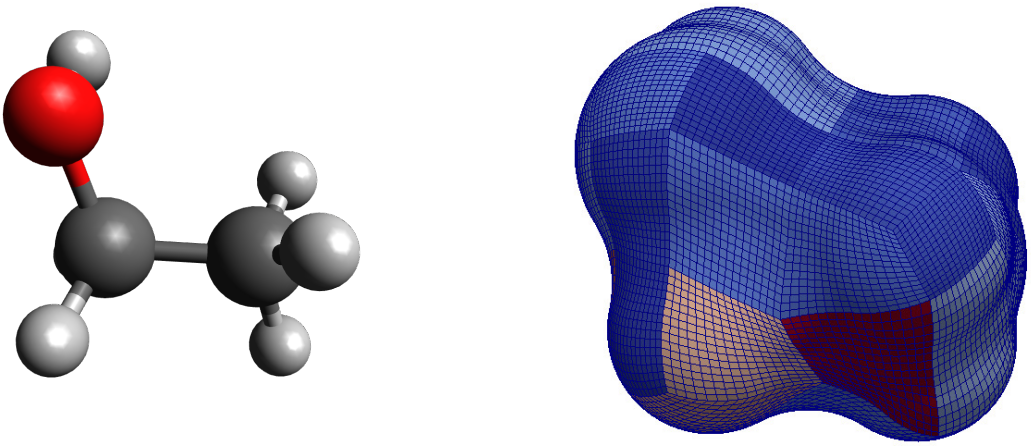


Figure 5.4. Ball and stick description of ethyl alcohol and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 40$, $fit = 0.43$.

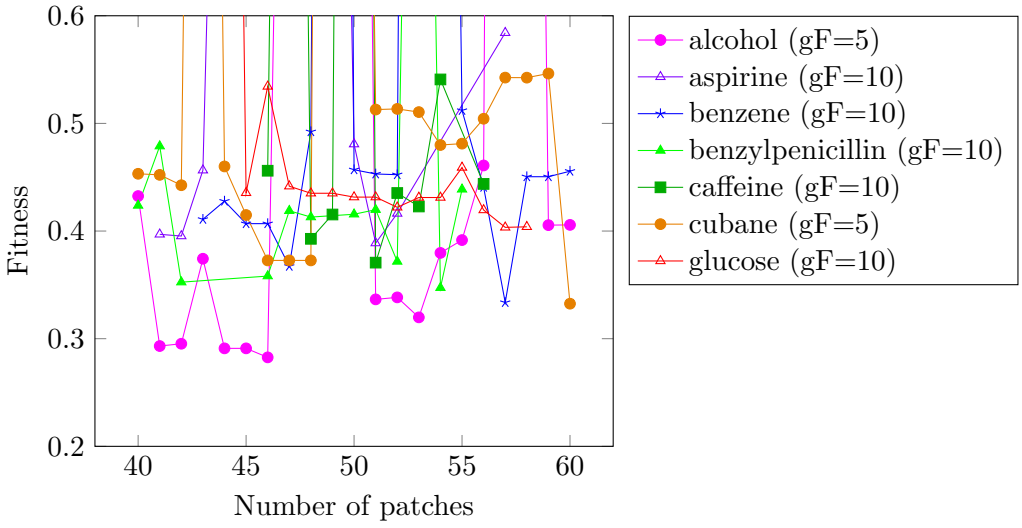


Figure 5.5. The worst node fitness versus the number of patches for the set of molecules calculated on level 4.

(■) point in time recorded is after determining the cubes situated on the surface. The second (■) point in time marks the computation of the surface points by means of the bisection algorithm seen in algo. 4.16. This time period depends on the number of surface cubes, which in turn also factors in the constant `geometryFactor`. The time needed to triangulate the resulting unstructured mesh is negligible and is thus not taken into account. The third (■) point in time recorded is the time needed to compute the fine mesh by means of edge replacement, as seen in algo. 4.22. The fourth (■) point in time is closely related to the third and marks the computation of the coarse mesh. The fifth (■) point in time pinpoints the time needed for the refinement up to the optimization level `patchOptimize`. The patch initialization and refinement towards the boundary that will be used for the improvement strategy are stored in the sixth (■) point in time. The seventh (■) and the eighth (■) point in time mark the time needed for the global and the local improvement strategies, respectively. The last and ninth (■) point in time contains the shift back to the original coordinates, the projection onto the surface with accuracy `tol_end`, the refinement up to the required level, n_L, and the output of the grid on all relevant levels in a format that can be read by the wavelet BEM code.

The time needed for the refinement and the output depends on the number of patches and the time needed to project the points onto the surface. This is a balancing act: if the number of patches is small, then the approximation of the surface is poorer and the time needed to project the refined patches onto the surface might take longer. On the other hand, the approximation of the surface is better if the number of patches is larger, but more points need to be projected onto the surface. Fig. 5.6 contains a representation of the time needed for the ethyl alcohol molecule. The pie chart on the left shows the time distribution for the automatic patch detection. The total time there is 116 sec. On the right the time for a fixed number of patches varies between 150 sec and 220 sec. For the other test molecules, please see figures A.3, A.6, A.9, A.12, A.15 and A.18.

The conclusion that can be reached is that most of the time is spent in the edge replacement algorithm and the final refinement and output steps. While the output cannot be optimized, a better, coarser starting mesh would save a lot of time currently spent in the edge replacement algorithm. The starting mesh, however, has to resolve all the fine structures of the geometry. This is not guaranteed with a larger cube size for the marching cubes algorithm.

5.1.3. Convergence of the wavelet BEM solver

In order to test the wavelet BEM solver, meshes on levels 2, 3, 4, 5, and 6 have been generated. The wavelet BEM solver was first used to solve the boundary integral equation in eq. (1.31) on these meshes. The results are shown in fig. 5.7 for piecewise constant ansatz functions and in fig. 5.8 for piecewise bilinear ansatz functions. In this configuration, the wavelet BEM code parameters are chosen as $a = 2$, $d' = 1.25$ and $d' = 2.25$ for the constant ansatz and bilinear ansatz functions, respectively, and $b = 10^{-3}$. For the iterative solver, the precision $\epsilon = 10^{-6}$ was taken. The figures show the difference in the ASC compared to the expected theoretical value described in eq. (5.1). Both

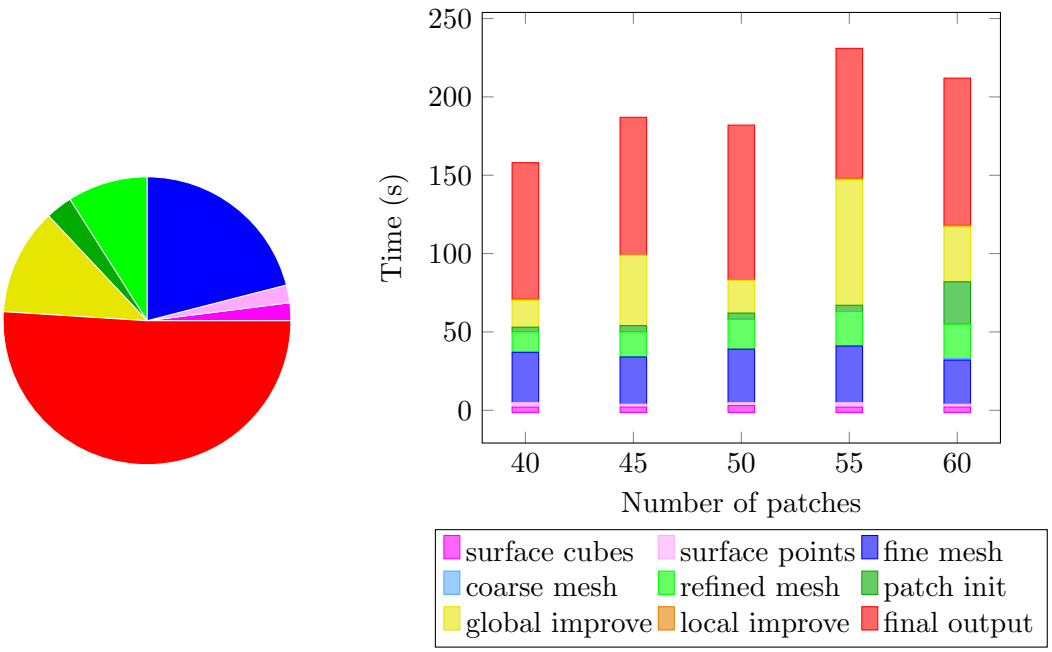


Figure 5.6. Time distribution for ethyl alcohol: On the left, the time for the automatic patch number detection can be seen, total time = 116 sec. On the right the version for fixed number of patches is depicted.

choices of ansatz functions converge for all molecules tested, starting with errors in the region of 10^{-1} and reaching up to 10^{-3} for the piecewise constant ansatz functions and reaching errors of around 10^{-5} for the piecewise bilinear ansatz functions. Hence, the piecewise bilinear ansatz functions perform better than the piecewise constant ansatz functions.

Secondly, the solution of the eq. (1.32) using the wavelet BEM solver was analyzed in fig. 5.9 for the piecewise constant ansatz functions and in fig. 5.10 for piecewise bilinear ansatz functions. The compression parameters were chosen the same as before, namely $a = 2$, $d' = 1.25$ and $d' = 2.25$ for the piecewise constant and bilinear ansatz functions, respectively, and $b = 10^{-3}$. For the GMRES solver, the precision $\epsilon = 10^{-6}$ was taken.

Again, we can see that the bilinear ansatz functions perform better, having errors between 10^{-1} and 10^{-4} compared to the constant ansatz functions that only reach a precision of about 10^{-1}. Moreover, one can see that the application of the wavelet BEM solver to eq. (1.31) performs better than on eq. (1.32).

For a more detailed analysis of the convergence of the wavelet BEM solver for the PCM, we refer to [5].

5.1.4. Time versus atoms

A simple scalability test that can be done for the cavity generator is to take a look at the polyalkane chains C_nH_{2n+2}, $n = 4, 8, 16, 32$. We look at the time needed to generate the

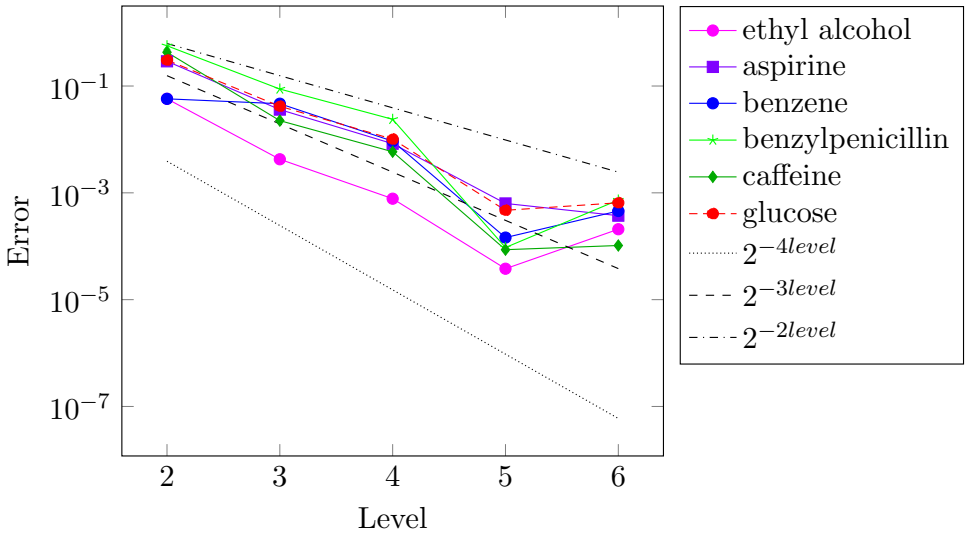


Figure 5.7. Convergence of the wavelet BEM solver for the standard PCM case found in eq. (1.31) for the definition of the surface using position and radii of atoms using piecewise constant ansatz functions, calculated as a difference to the theoretical value.

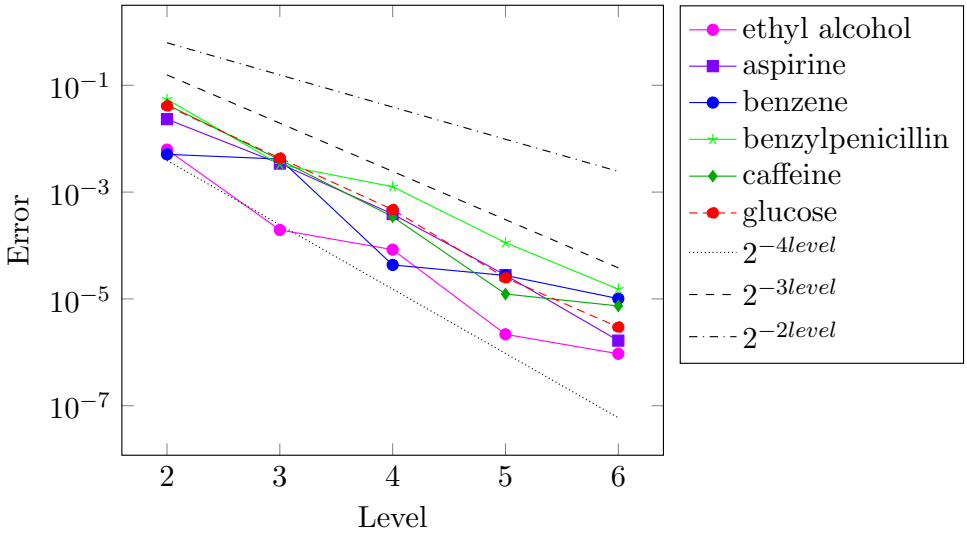


Figure 5.8. Convergence of the wavelet BEM solver for the standard PCM case found in eq. (1.31) for the definition of the surface using position and radii of atoms using piecewise bilinear ansatz functions, calculated as a difference to the theoretical value.

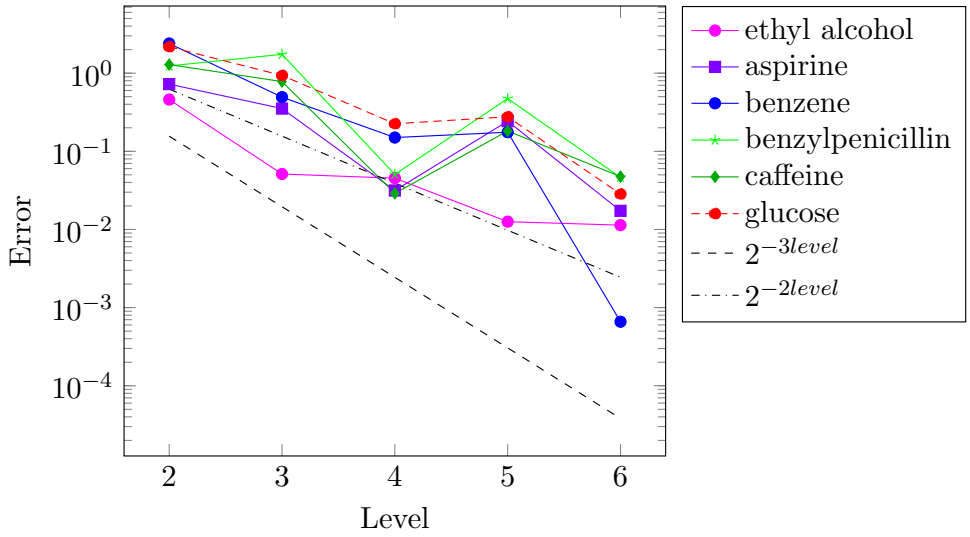


Figure 5.9. Convergence of the wavelet BEM solver for the standard PCM case found in eq. (1.32) for the definition of the surface using position and radii of atoms using piecewise constant ansatz functions, calculated as a difference to the theoretical value.

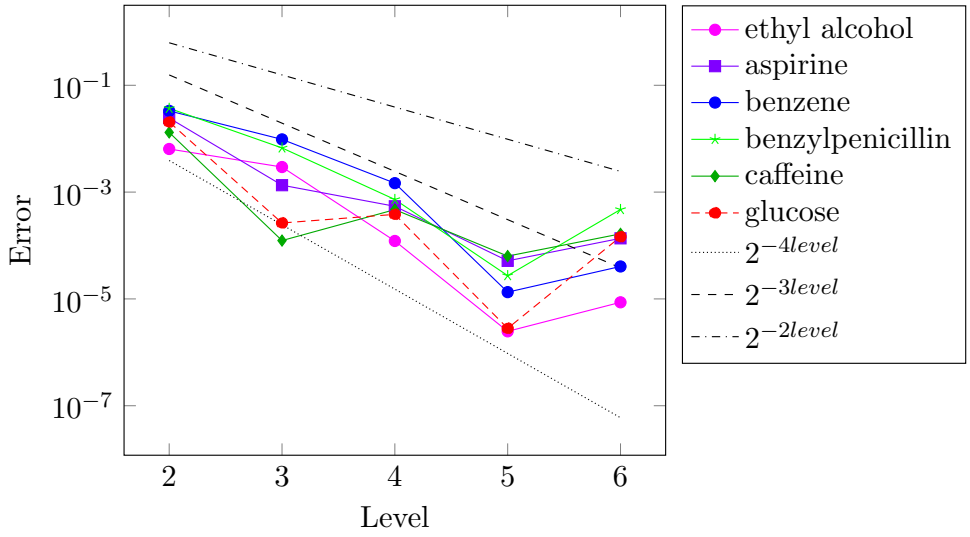


Figure 5.10. Convergence of the wavelet BEM solver for the standard PCM case found in eq. (1.32) for the definition of the surface using position and radii of atoms using piecewise bilinear ansatz functions, calculated as a difference to the theoretical value.

mesh using an automatic patch number detection with the resulting number of patches $n_{\square}(C_4H_{10}) = 389$, $n_{\square}(C_8H_{18}) = 399$, $n_{\square}(C_{16}H_{34}) = 603$, and $n_{\square}(C_{32}H_{66}) = 1999$. The patches that have been generated, coloured corresponding to their fitness value, are shown on the left side in fig. 5.11. On the right side, the pie chart of the time distribution for the automatic patch detection on refinement level 4 is shown. One can see that the time required for each step is similar for every molecule.

(a) C_4H_{10} patch representation with automatic patch detection, $n_{\square} = 389$, $fit = 0.58$.

(b) C_8H_{18} patch representation with automatic patch detection, $n_{\square} = 399$, $fit = 3.6$.

(c) $C_{16}H_{34}$ patch representation with automatic patch detection, $n_{\square} = 603$, $fit = 3.4$.

(d) $C_{32}H_{66}$ patch representation with automatic patch detection, $n_{\square} = 1999$, $fit = 1.56$.

Figure 5.11. Surface description with automatic number of patch detection.

The blue line in the logarithmic plot found in fig. 5.12 depicts the total time needed for the run, while the black line represents the total time divided by the number of patches used. The black line shows that the time needed per patch is proportional to the number of carbon atoms found in the molecule under consideration. Nonetheless, the test does not show linear scalability in the total time needed because the number of patches that were automatically detected does not scale with the size of the molecule.

5.2. Geometry description by isosurfaces

As mentioned in chpt. 4, the only information needed by the mesh generator is a characteristic function. As used in the IPCM version implemented Gaussian, [19], the charge distribution in space can be used to find the surface of the molecule. The isosurface is a set of points that have the same value, which in this case means the same charge value. The surface that will eventually be used as a description of the molecular cavity is then given by the isosurface with a certain charge value. In our example, the value 0.02 has been chosen.

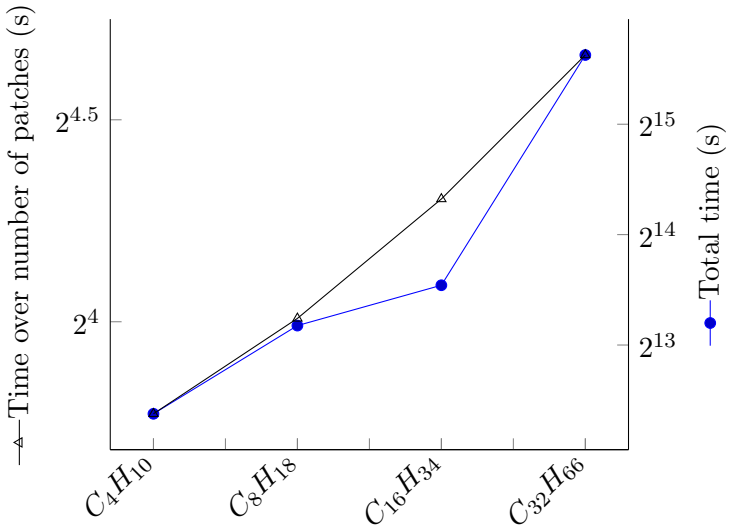


Figure 5.12. Time analysis for the polyalkane chains, C_nH_{2n+2}, $(n = 4, 8, 16, 32)$, $n_\square(C_4H_{10}) = 389$, $n_\square(C_8H_{18}) = 399$, $n_\square(C_{16}H_{34}) = 603$, $n_\square(C_{32}H_{66}) = 1999$.

5.2.1. Input file of the isosurface

The benzene molecule is used to prove the applicability of this approach. The input file was generated by Gaussian09, using Hartree-Fock and the 6-311++G** basis set. The structure of the input file generated is shown in lst. 5.13. The first two lines are comments and are ignored during parsing. The third line contains the number of atoms and the position of the centre of the bounding box. The next three lines contain the number of cells and the axis vectors for the x-, y-, and z-direction. The sizes are given in Bohr if the number of the cells is positive, while the sizes are given Ångström if the number of the cells is a negative number. In our version of the file, the sizes are given in Bohr, which also means that the charge values for the cubes are given in Bohr^{-3}. The current version of the code works only when the bounding box is parallel to the axis and defined by linearly independent vectors. The lengths of the axes defining the bounding box are proportional to the size of the cells in each direction. The last part of the header contains a line for each atom including the atom number, the atomic charge and the x-, y-, and z-position of the centre of the atoms. These lines are used to generate the charge input file for the wavelet BEM code. In order to achieve a resulting mesh with respect to these positions of the atoms, the mesh needs to be translated at the end in the opposite direction of the centre of the bounding box. The following lines contain the value at the each voxel.

The main function that needs to be adapted is the method checking whether a point is inside or outside the molecule. A point is inside the molecule if the value of the interpolated voxels evaluated at the given point is larger than the constant `isoSurfaceValue`. We have chosen `isoSurfaceValue` = 0.02 for our initial test.

A cubic Newton interpolation is used, placing the requested point in the middle voxel

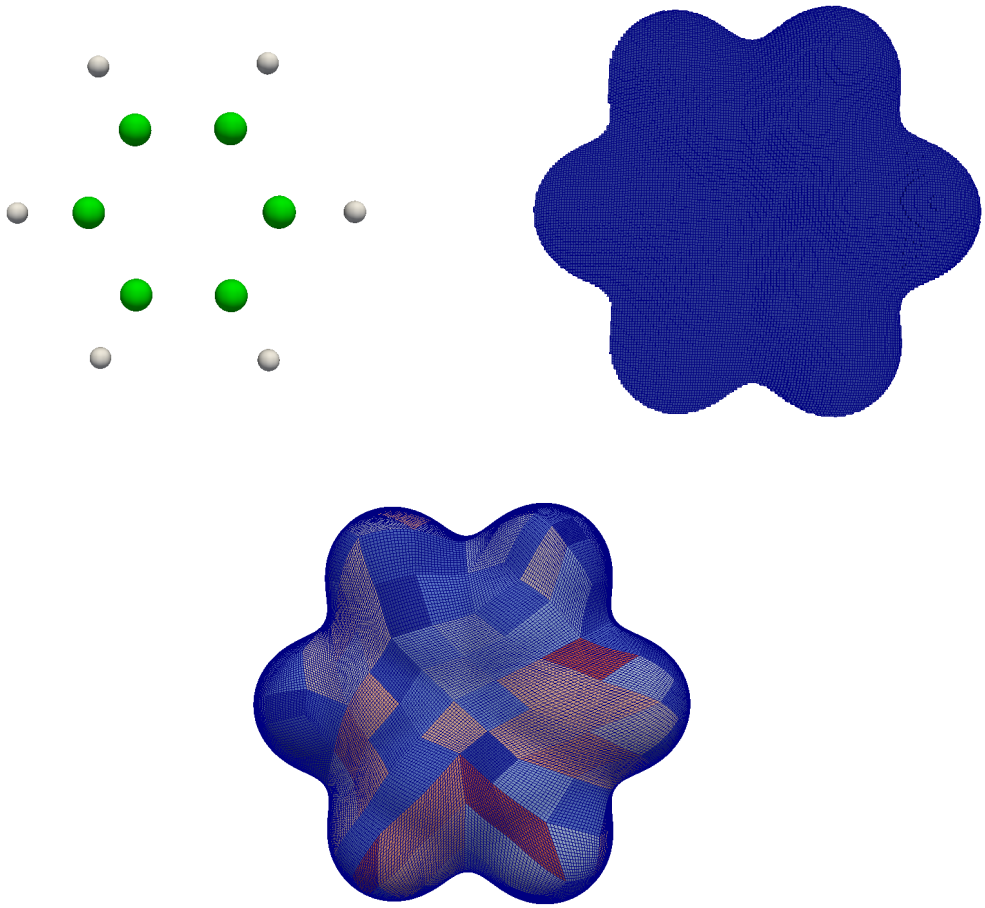


Figure 5.14. The atom information as found in the cube file, the isosurface for `isoSurfaceValue = 0.02` extracted directly from the cube file and the mesh generated with automatic patch number detection, $n_{\square} = 243$ patches on level 4, $fit = 0.6$.

Listing 5.13 Input file for isosurface calculations.

```
Hartree-Fock in vacuo, benzene potential=scf
Electrostatic potential from Total SCF Density
  12    0.000000   -7.264863  -23.367220
 246    0.083333    0.000000    0.000000
 212    0.000000    0.083333    0.000000
 281    0.000000    0.000000    0.083333
   6    6.000000    9.966416    3.777563  -16.191174
   6    6.000000   12.523215    3.813467  -15.512762
   6    6.000000   13.919723    1.566583  -15.499534
   6    6.000000   12.759431   -0.716206  -16.164717
   6    6.000000   10.202631   -0.752111  -16.841239
   6    6.000000    8.806124    1.494773  -16.854467
   1    1.000000    8.889272    5.510441  -16.200622
   1    1.000000   13.418945    5.574692  -15.000646
   1    1.000000   15.892597    1.594929  -14.977969
   1    1.000000   13.836575   -2.449085  -16.153379
   1    1.000000    9.306901   -2.513336  -17.353355
   1    1.000000    6.833250    1.466427  -17.377922
 8.23514E-04  8.34400E-04  8.45257E-04  8.56080E-04
   8.66862E-04  8.77596E-04 ...
```

of the interpolant. The size of the marching cube algorithm is chosen to be a multiple of the voxel size found in the cube file, `step_cube`, as follows

$$\texttt{step} = \texttt{geometryFactor} * \texttt{step\_cube},$$

where `geometryFactor` is a constant and `step` and `step_cube` are vector values for each spatial direction. In our example, we have

$$\texttt{step\_cube} = \begin{pmatrix} 0.0833333 \\ 0.0833333 \\ 0.0833333 \end{pmatrix},$$

found in lines $4-6$ of the input file and we have chosen $\texttt{geometryFactor} = 4$. The resulting mesh for the benzene example using automatic patch detection can be found in fig. 5.14, where the information stored in the cube file is also shown. In the first subfigure the atom information extracted from the cube file is depicted. The second image illustrates the cubes in the file where the associated value is larger than the chosen `isoSurfaceValue = 0.02`. The time needed for this implementation is much smaller than for the description given by atoms as introduced in chpt. 4. This can be seen in fig. 5.15 where the y-scale is in the order of seconds. The test if one point is inside or outside the molecule is done much faster, which also leads to a different distribution of the time. The final point in time containing the refinement and the output is no longer dominant, now most of the time is spent in the edge replacement

algorithm, as seen in the fixed number of patches. For the automatic patch detection, more time is spent in the refinement step, maybe due to the fact that we have a large number of patches, namely $n_\square = 243$. For a more reliable conclusion more molecules would need to be examined.

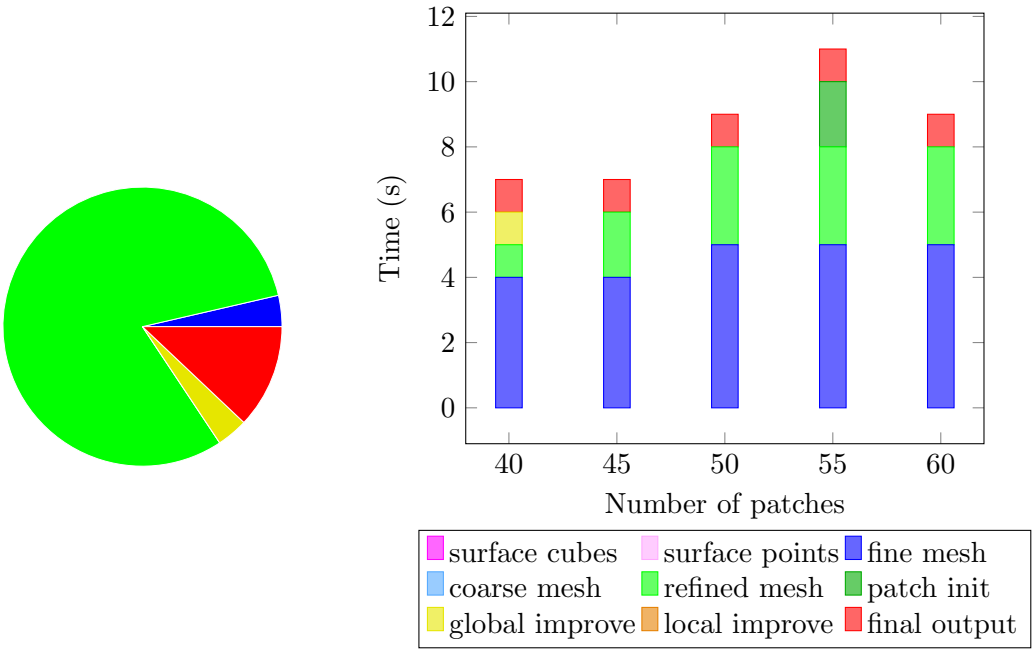


Figure 5.15. Time distribution for benzene described by isosurfaces. On the left the automatic patch number detection can be seen, total time = 39 sec and on the right the version for fixed number of patches is shown.

5.2.2. Convergence of the wavelet BEM solver

In this subsection, the convergence of the wavelet BEM solver on the mesh generated by isosurfaces is examined. To this end, meshes for benzene are generated for different levels in the same way as before. The resulting meshes are used as an input for the wavelet BEM solver for eq. (1.31) using piecewise constant and piecewise bilinear ansatz functions. The convergence behaviour is shown in fig. 5.16 for the parameter choice $a = 2$, $d' = 1.25$ for the piecewise constant ansatz functions and $d' = 2.25$ for the piecewise bilinear ansatz functions and $b = 10^{-3}$. For the iterative solver, the precision is $\epsilon = 10^{-6}$ taken. The overall convergence rate is seen to be similar to the one shown in fig. 5.7 and fig. 5.8 for the description given by atoms and radii. This shows that the cavity generator produces meshes of similar quality in both cases.

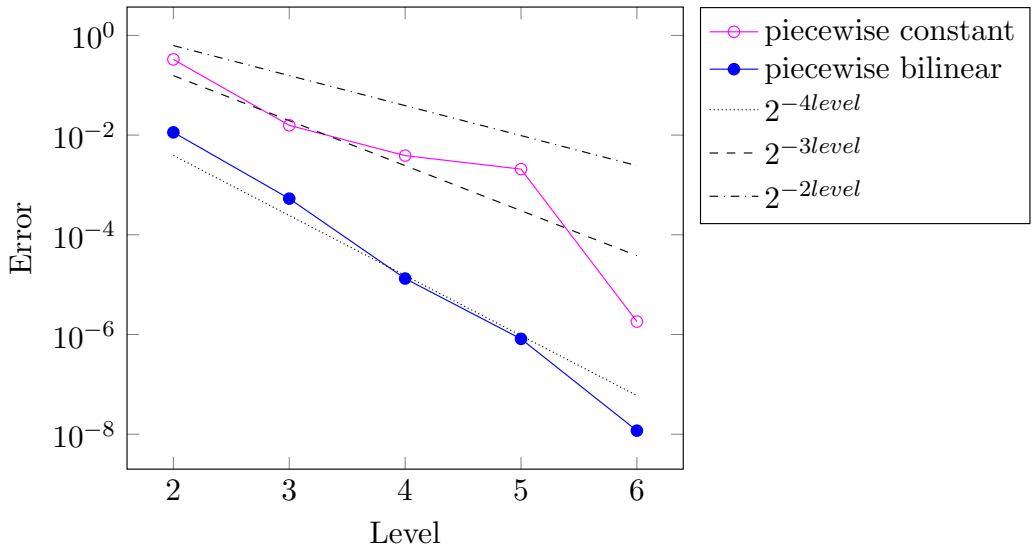


Figure 5.16. Convergence of the wavelet BEM solver for the standard PCM case for the definition of benzene by an isosurface with the isovalue `isoSurfaceValue = 0.02`.

5.3. PCM for ionic solutions

Applying the wavelet BEM solver to the ethyl alcohol surface under variation of κ in the range from 0 to 1 at several discretization levels reveals the influence of the ionic strength of the solution on the ground state energy. The results are depicted in fig. 5.17. An increase of the value for κ from 0 to 1 leads to a decrease in the ground state energy. This decrease is not linear and can be seen in fig. 5.18. Independently of the discretization level, the drop in ground state energy is much bigger at the beginning of the range, $\kappa \in [0, 1]$, compared to its end. The decrease in ground state energy can be rationalized by considering the polar nature of the ethanol molecule. The dipole moment of ethyl alcohol becomes more stable in a stronger ionic environment. Therefore, the ground state energy drops with κ.

The usage of PCM in the case of ionic solutions was described in [7]. Like in the original paper, the dielectric constant for the medium is chosen $\varepsilon = 78.5$ and the ionic screening constant is taken in the range $\kappa \in [0, 1]$, with 0 being normal water and 1 being a fully ionic liquid.

Tab. 5.19 shows the compression behaviour for different values of κ. The *a priori* compression of both, the interior and the exterior boundary integral operators, and the *a posteriori* compression of the interior boundary integral operators are the same for all values of κ. For the exterior boundary operators, the *a posteriori* compression shows a slight decrease in the number of non-zero elements in the system matrices for larger values of κ. One can notice also a slight increase in the non-zero elements of the exterior boundary integral operators compared to the interior ones, for all values of κ. This test demonstrates that the wavelet BEM solver can also be applied in the case of ionic solutions. Further investigation and validation is however needed.

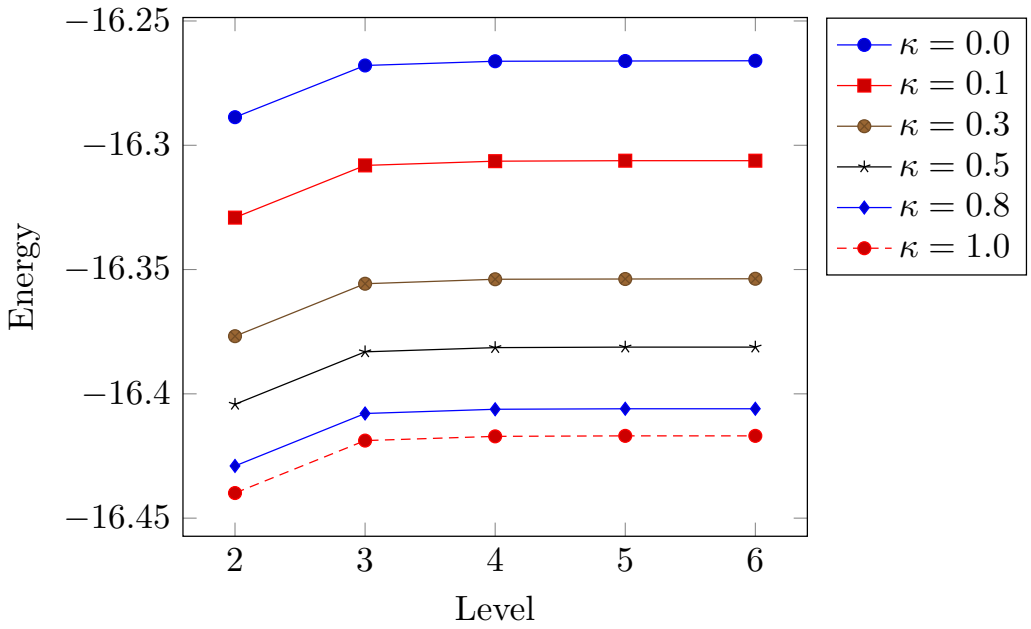


Figure 5.17. Ground state energy for ionic solutions with $\varepsilon = 78.5$ as introduced in [7] using piecewise bilinear wavelets on the surface of ethyl alcohol depending on the level of refinement.

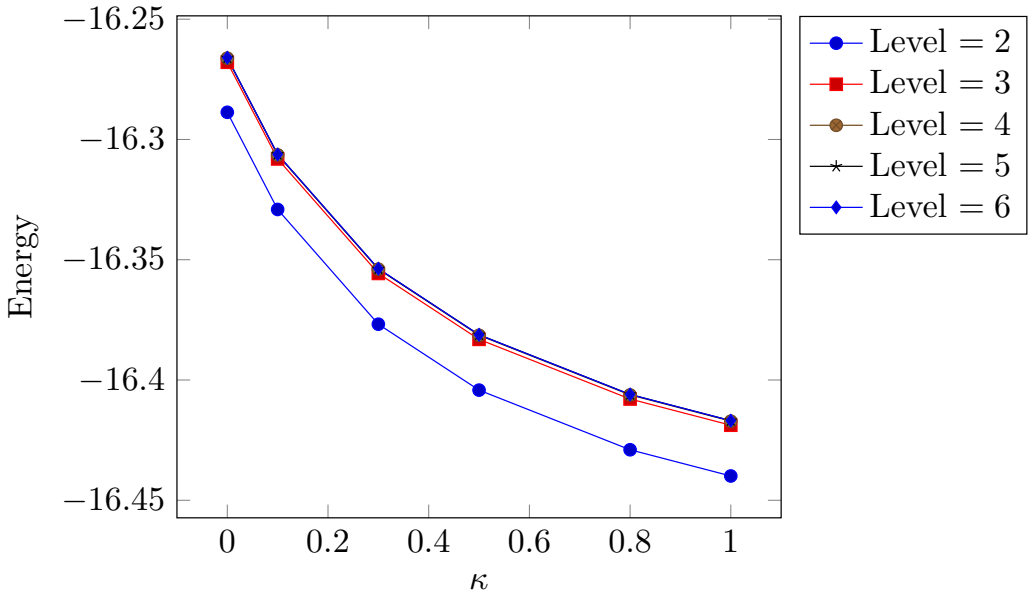


Figure 5.18. Ground state energy for ionic solutions with $\varepsilon = 78.5$ as introduced in [7] using piecewise bilinear wavelets on the surface of ethyl alcohol depending on the ionization constant.

Table 5.19. Effect of the ionic screening constant κ on the *a priori* and *a posteriori* matrix compression for ethyl alcohol for piecewise bilinear ansatz functions. The compression parameters are set to the values $a = 1$, $d' = 2$, and $b = 0.001$.

level	*a priori* compression (int/ext)	*a posteriori* compression (interior)	*a posteriori* compression (exterior) $\kappa = 0.0$	$\kappa = 0.1$	$\kappa = 0.3$	$\kappa = 0.5$	$\kappa = 0.8$	$\kappa = 1.0$
2	27.16310 %	24.87055 %	24.86752 %	24.85800 %	24.82586 %	24.79410 %	24.72091 %	24.65369 %
3	13.39761 %	10.37040 %	10.36739 %	10.36096 %	10.34199 %	10.31283 %	10.25253 %	10.19618 %
4	4.39581 %	3.25333 %	3.25243 %	3.25196 %	3.24790 %	3.24113 %	3.22563 %	3.21189 %
5	1.27997 %	0.95150 %	0.95135 %	0.95126 %	0.95056 %	0.94926 %	0.94637 %	0.94363 %
6	0.34789 %	0.26583 %	0.26580 %	0.26578 %	0.26567 %	0.26546 %	0.26498 %	0.26451 %

5.4. PCM for liquid crystals

The last case that the PCM can has been tested on is the case of liquid crystals, as described in [6, 7]. The subject of this paragraph is to show that also the wavelet BEM implementation of the PCM is applicable in this situation. To this end, the dielectric tensor for the liquid crystal 7CB (4-*n*-heptyl-4'-cyanobiphenyl) given by

$$\boldsymbol{\varepsilon} = \begin{pmatrix} 5.54 & 0 & 0 \\ 0 & 5.54 & 0 \\ 0 & 0 & 17.1 \end{pmatrix} \tag{5.20}$$

has been used. The test molecule chosen is ethyl alcohol and the ground state energy in the liquid crystal solution of ethyl alcohol has been compared to the ground state energy in the isotropic medium characterized by the dielectric constant given by the maximum, $\varepsilon_{\max} = 17.1$, and the minimum, $\varepsilon_{\min} = 5.54$, found in the tensor. The values for this comparison can be found in tab. 5.22. They show that the energy in the crystal is situated in between the two values of the isotropic mediums. Fig. 5.21 compares the convergence rates of the piecewise constant and the piecewise bilinear ansatz functions compared to the energy value calculated on refinement level 6 when using piecewise bilinear ansatz functions. The compression parameter chosen are $a = 2$, $d' = 1.25$ for the piecewise constant ansatz functions and $d' = 2.25$ for the bilinear case, respectively, and $b = 10^{-3}$. The precision of the iterative solver was again taken $\epsilon = 10^{-6}$.

Tab. 5.23 points out the compression rates found in this case. The *a posteriori* compression of the interior and the exterior operator are proven to be of comparable size. This time, however, the exterior boundary integral operator is a little denser than the interior one. This little test shows that the wavelet BEM solver can be applied also in the case of liquid crystals and yields results that behave like the ones described in [7]. Further investigation is required in order to compare the wavelet BEM solver with different PCM codes in order to examine the performance and accuracy.

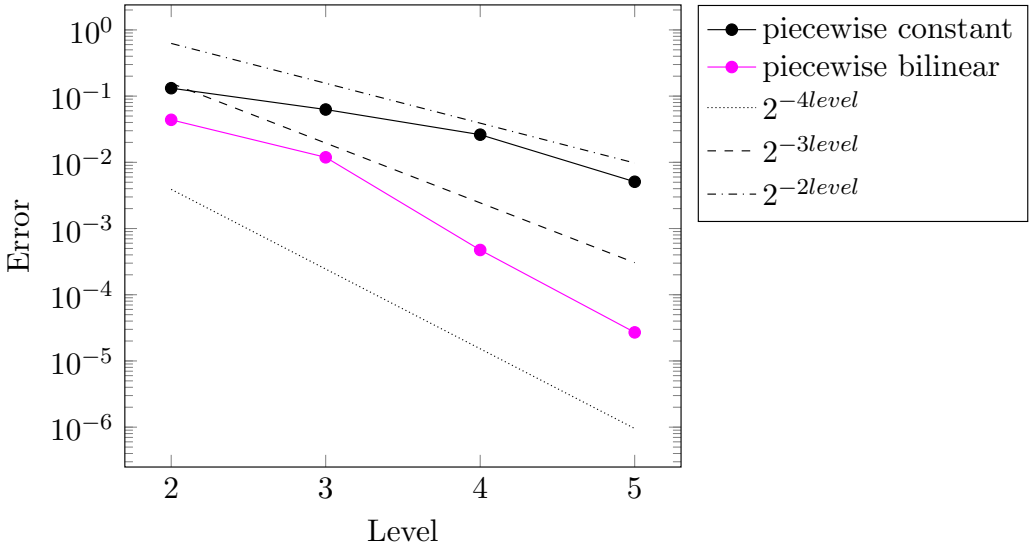


Figure 5.21. Convergence of the ground state energy of ethyl alcohol for the liquid crystal case of 7CB with the dielectric tensor given by $\boldsymbol{\varepsilon} = \mathrm{diag}(5.54, 5.54, 17.1)$ as described in [7] as a difference to the value calculated on the discretization level 6.

Table 5.22. The ground state energy for ethyl alcohol for piecewise constant ansatz functions and piecewise bilinear ansatz functions in the liquid crystal solution with the dielectric tensor given by $\boldsymbol{\varepsilon} = \mathrm{diag}(5.54, 5.54, 17.1)$. Comparison to two isotropic dielectric solutions described by $\varepsilon_{\min} = 5.54$ and $\varepsilon_{\max} = 17.1$ as in [7].

level	constant $\varepsilon_{\min} = 5.54$	bilinear $\varepsilon_{\min} = 5.54$	constant	bilinear	bilinear $\varepsilon_{\max} = 17.1$	constant $\varepsilon_{\max} = 17.1$
2	59.5972	59.5493	64.2534	64.2095	68.8066	68.7634
3	59.5655	59.5401	64.2189	64.1935	68.7689	68.7444
4	59.5474	59.5350	64.2007	64.1878	68.7520	68.7390
5	59.5375	59.5349	64.1901	64.1876	68.7412	68.7387
6	59.5349	59.5349	64.1876	64.1876	68.7387	68.7387

Table 5.23. The *a priori* and *a posteriori* matrix compression for ethyl alcohol for piecewise bilinear ansatz functions in a liquid crystal solution. The compression parameters are set to the values $a = 1$, $d' = 2$, and $b = 0.001$.

level	*a priori* compression (int/ext)	*a posteriori* compression (interior)	*a posteriori* compression (exterior)
2	27.16310 %	24.87055 %	24.97164 %
3	13.39761 %	10.37040 %	10.46560 %
4	4.39581 %	3.25333 %	3.30614 %
5	1.27997 %	0.95150 %	0.96576 %
6	0.34789 %	0.26583 %	0.26903 %

CHAPTER 6

Conclusions

In chemistry and nature, molecules often reside in solutions. Thus, simulating the solute-solvent interaction has become a very important field of research. One method, the polarizable continuum model, discards the molecular structure of the solvent entirely, reducing the interaction to the boundary of the solute molecule, the cavity surface. The drawbacks of implementations using standard boundary element ansatz functions are the dependency on an accurate description of the molecular surface and the limitation due to the resulting dense system matrices.

A new C++ implementation of the wavelet boundary element method has been developed, which shows flexibility in the type of ansatz functions that can be used. The advantages of the wavelet method stem from the quasi-sparsity of the system matrices which allow the use of finer discretization, solving larger problems or at a higher accuracy. C++ has been chosen due to its class structures and template variables and functions.

Since wavelets are defined as a tensor product of one dimensional functions, a new mesh generator was designed that discretizes the surface into uniform quadrangular patches. In order to calculate a surface mesh, it needs only a characteristic function or a level set function. The description of the molecular surface can be given by atoms or, alternatively, the isodensity of the charge distribution can be used as a level set function. This makes it a very versatile tool for generating meshes on any smooth surfaces.

The mesh generator starts by using the marching cubes algorithm to find an initial description of the surface. Several types of fitness functions quantify the quality of the mesh at different points in time: the fitness functions for triangles, the fitness function used to calculate the triangle pairs for the patches, and the fitness function of the quadrangular patch used for the mesh improvement. An intermediate step generates a triangular mesh of good quality with planar triangles that could also be used with

standard boundary element method codes, especially with fast multipole methods.

The validation of the mesh generator was done by examining seven test molecules and using the wavelet boundary element method to calculate the difference in the apparent surface charge. The three different cases of the polarizable continuum model have also been implemented, namely the cases of neutral solutions, of ionic solutions, and of liquid crystals. Therefore, this work opens the gates towards larger systems that can be computed with tremendous accuracy.

"*The reward of work well done is the opportunity to do more.*" – Jonas Salk. Replacing the marching cubes algorithm by a well distributed point cloud that resolves the geometry without superfluous points can speed up the computation. Secondly, the final meshes could be improved by finding fitness functions that take into account the curvature and the resulting quadrangles. Finally, the wavelet boundary element method can be extended to geometrical updates, by changing the position of the atoms under the influence of the solution compared to the initial position in vacuum. In each step, the surface gradient needs to be computed, the atoms need to be moved, and a new surface mesh has to be generated.

APPENDIX A

Molecules used

Listing A.1 Input file aspirine.

```
21
-6.96612        1.50927         0.10692        1.70
-5.55275        1.52902         0.04836        1.70
-4.88412        2.78054        -0.01210        1.70
-5.64356        3.96767        -0.02855        1.70
-7.03583        3.92328         0.02026        1.70
-7.69394        2.69942         0.09045        1.70
-3.39396        2.86991        -0.06364        1.70
-4.81062        0.38411         0.07174        1.52
-5.19277       -0.89694        -0.18920        1.70
-4.24048       -2.00971         0.11028        1.70
-3.29964       -1.85424        -0.45793        1.20
-4.67564       -2.99128        -0.17544        1.20
-4.01317       -2.01950         1.19656        1.20
-2.63328        1.75371        -0.05298        1.52
-2.84817        3.96308        -0.11641        1.52
-6.26839       -1.14991        -0.70972        1.52
-1.66736        1.80415        -0.08619        1.20
-7.51833        0.58665         0.18034        1.20
-8.77513        2.67144         0.13679        1.20
-5.15620        4.93346        -0.07815        1.20
-7.60639        4.84309         0.00862        1.20
```

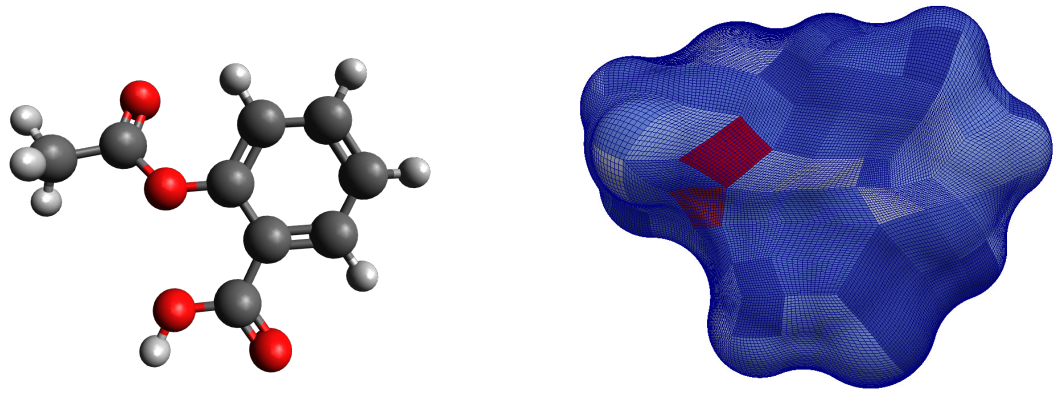


Figure A.2. Ball and stick description of aspirine and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 159$, $fit = 2.26$.

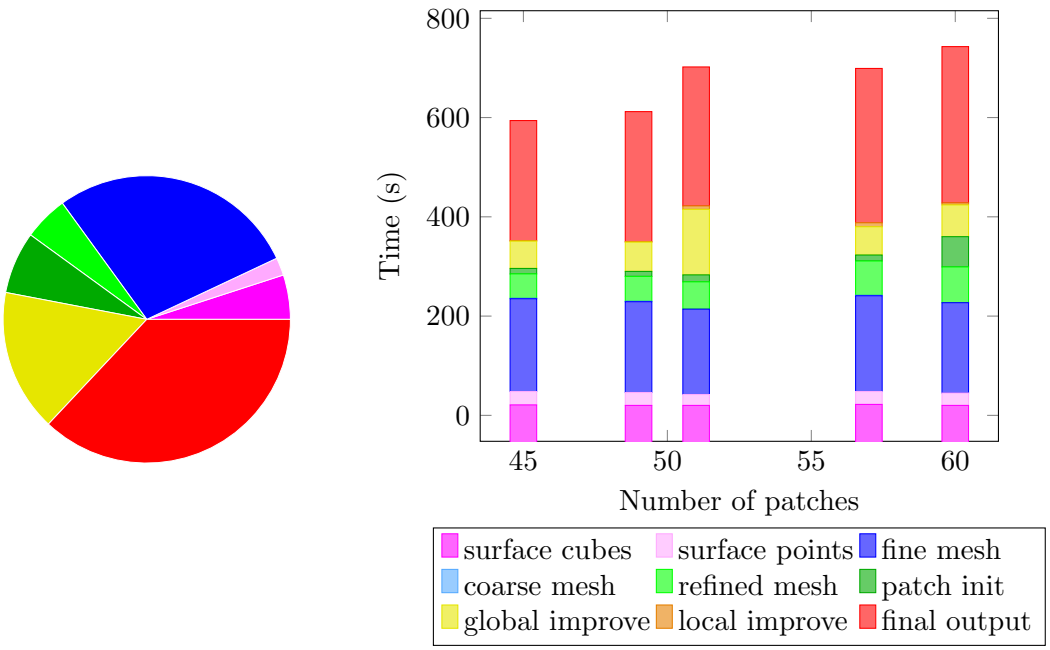


Figure A.3. Time distribution for aspirine. On the left the automatic patch number detection can be seen, total time = 2644 sec. On the right the version for fixed number of patches is depicted.

Listing A.4 Input file benzene.

```
12
 1.212        -0.700         0.002        1.7
 1.212         0.700         0.002        1.7
 0.000         1.400         0.000        1.7
-1.212         0.700        -0.002        1.7
-1.212        -0.700        -0.002        1.7
 0.000        -1.400         0.000        1.7
 2.148        -1.240         0.003        1.2
 2.148         1.240         0.003        1.2
 0.000         2.480         0.000        1.2
-2.148         1.240        -0.003        1.2
-2.148        -1.240        -0.003        1.2
 0.000        -2.480         0.000        1.2
```

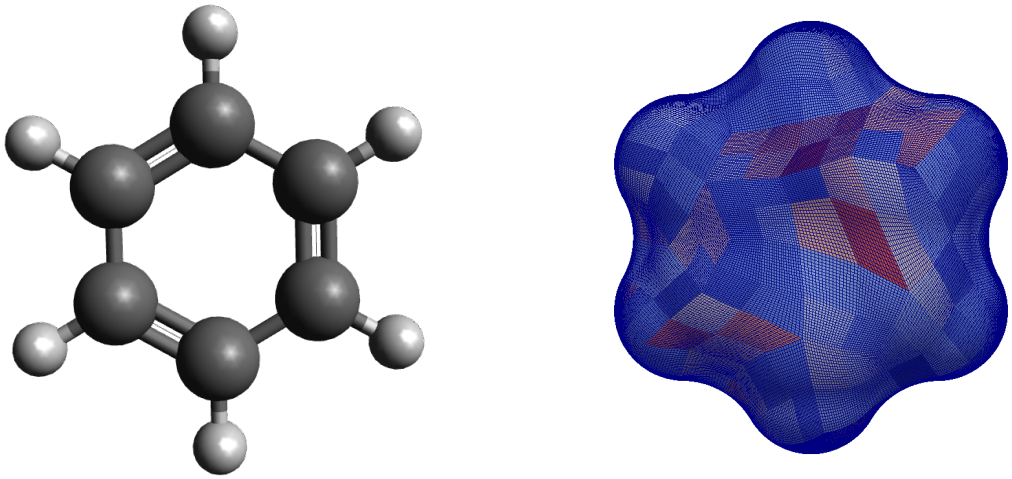


Figure A.5. Ball and stick description of benzene and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 327$, $fit = 0.457338$.

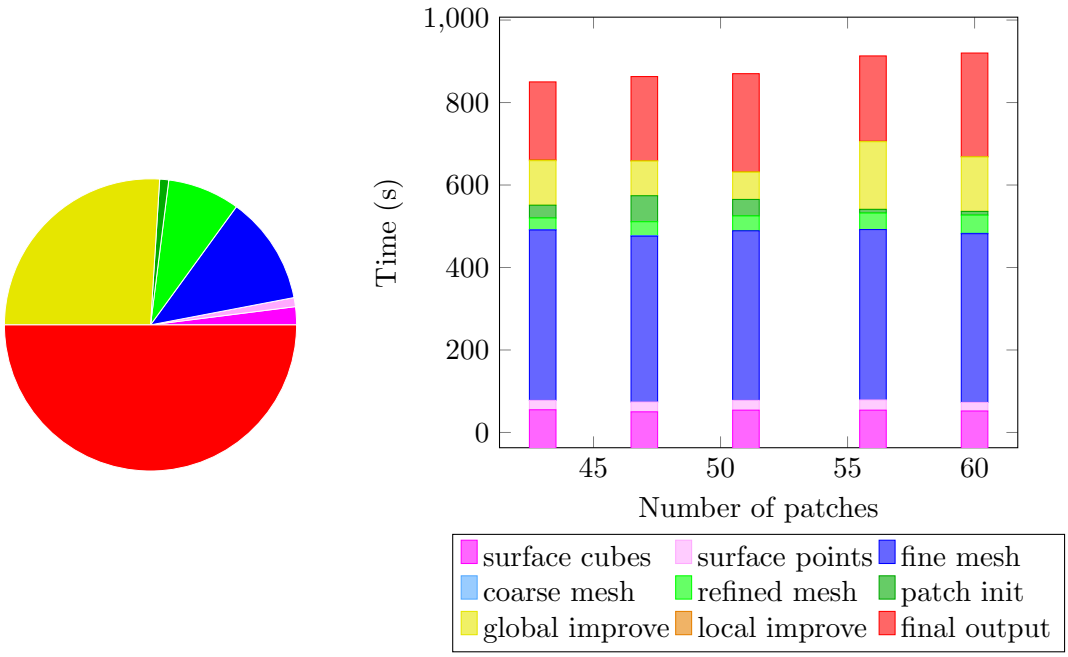


Figure A.6. Time distribution for benzene. On the left the automatic patch number detection can be seen, total time = 2321 sec. On the right the version for fixed number of patches is depicted.

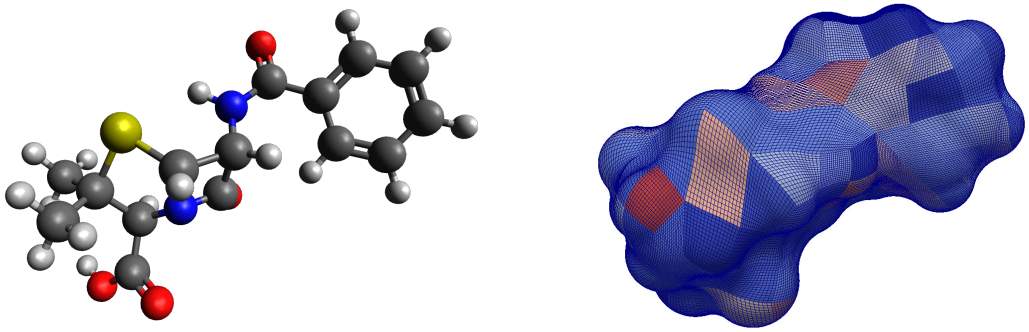


Figure A.8. Ball and stick description of benzylpenicillin and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 159$, $fit = 0.614$.

Listing A.7 Input file benzylpenicillin.

```
38
-4.34515        0.69991        0.54911        1.70
-3.22398        1.58638       -0.22539        1.70
-2.47684        0.38611       -0.38603        1.55
-3.33068       -0.35661        0.27724        1.70
-2.10750        2.67551        0.74049        1.80
-0.64739        1.86379       -0.02753        1.70
-1.04227        0.36386       -0.21197        1.70
-0.38902        2.58902       -1.36621        1.70
 0.57397        2.02571        0.89221        1.70
-3.03351       -1.32487        0.96272        1.52
-0.36325       -0.33991       -1.37609        1.70
-1.02969       -0.73981       -2.31870        1.52
 0.96825       -0.55992       -1.37798        1.52
-5.28030        0.64726       -0.04650        1.20
 0.38870        1.54011        1.87419        1.20
 0.77702        3.10258        1.07510        1.20
 1.48613        1.58789        0.44477        1.20
-1.26314        2.50260       -2.04520        1.20
 0.49725        2.16499       -1.88344        1.20
-0.19228        3.66844       -1.18910        1.20
-3.55913        1.97887       -1.21016        1.20
-4.49467        0.98071        1.96880        1.55
-0.81835       -0.21344        0.71383        1.20
 1.52699       -0.32044       -0.62941        1.20
-3.78590        1.57429        2.42070        1.20
-5.58908        0.64879        2.72559        1.70
-5.65526        1.11105        3.85656        1.52
-6.73977       -0.21368        2.26402        1.70
-6.63756       -1.18396        1.23969        1.70
-7.74505       -1.95604        0.87490        1.70
-5.71437       -1.38323        0.73199        1.20
-8.96740       -1.78372        1.52181        1.70
-7.65080       -2.69670        0.09074        1.20
-9.08521       -0.84659        2.54617        1.70
-9.82142       -2.38363        1.23506        1.20
-7.98290       -0.07384        2.92114        1.70
-10.03280      -0.71754        3.05342        1.20
-8.10522        0.64764        3.71969        1.20
```

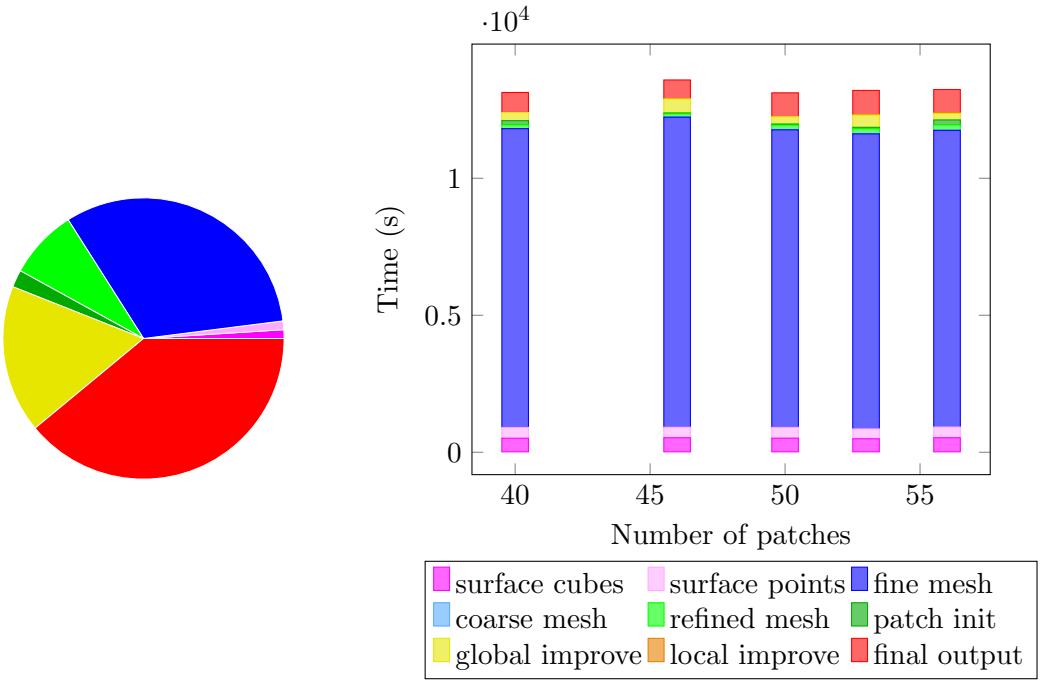


Figure A.9. Time distribution for benzylpenicillin. On the left the automatic patch number detection can be seen, total time = 6022 sec. On the right the version for fixed number of patches is depicted.

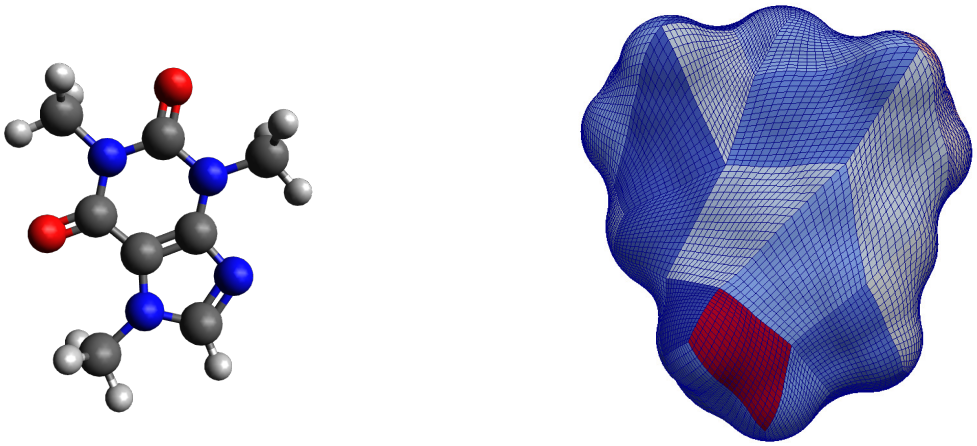


Figure A.11. Ball and stick description of caffeine and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 44$, $fit = 2.13$.

Listing A.10 Input file caffeine.

```
24
-2.58577        -0.23840         0.00000        1.70
-2.29749         1.08599        -0.00000        1.55
-0.94360         1.15140        -0.00000        1.70
-0.45954        -0.11567         0.00000        1.70
-1.47055        -1.00380         0.00000        1.55
 0.86612        -0.40408         0.00000        1.55
-0.05635         2.23540        -0.00000        1.70
 1.31478         1.96725        -0.00000        1.55
 1.76669         0.64775        -0.00000        1.70
-0.47482         3.38138        -0.00000        1.52
 2.96659         0.41484        -0.00000        1.52
 2.32343         3.04343        -0.00000        1.70
-3.30988         2.14267        -0.00000        1.70
 1.34423        -1.79146         0.00000        1.70
 1.87886         4.05804        -0.00000        1.20
 2.97197         2.94802        -0.89958        1.20
 2.97197         2.94802         0.89958        1.20
 0.50914        -2.52245         0.00000        1.20
 1.96035        -1.97196         0.90593        1.20
 1.96035        -1.97196        -0.90593        1.20
-3.58881        -0.64435         0.00000        1.20
-4.32908         1.68905        -0.00000        1.20
-3.22225         2.76766        -0.91148        1.20
-3.22225         2.76766         0.91148        1.20
```

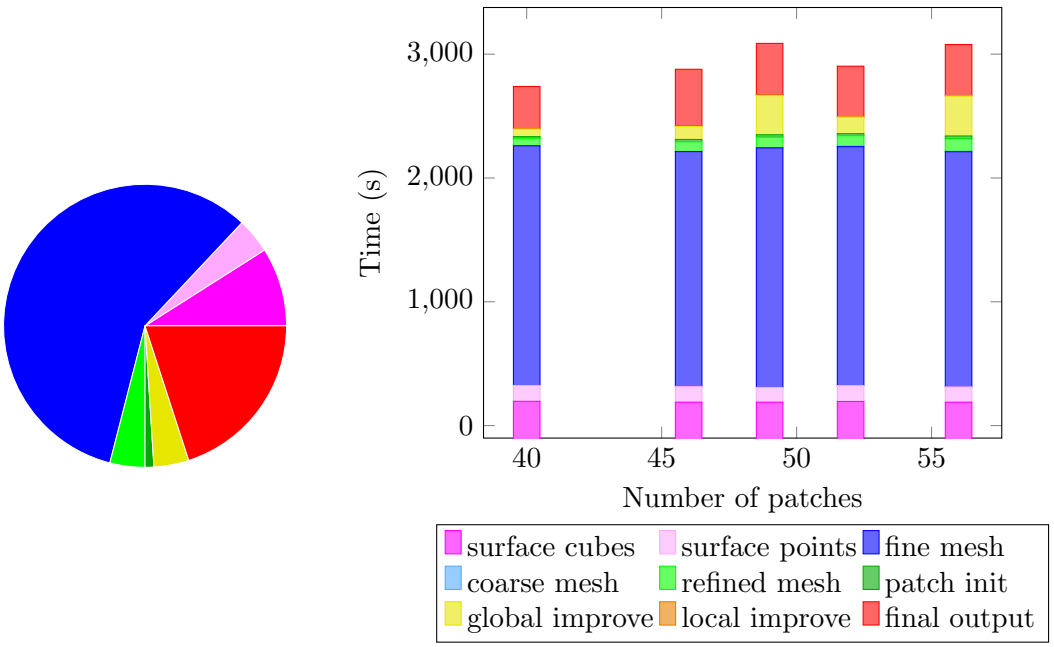


Figure A.12. Time distribution for caffeine. On the left the automatic patch number detection can be seen, total time = 1899 sec. On the right the version for fixed number of patches is depicted.

Listing A.13 Input file cubane.

```
16
-7.39867        -0.22073         0.21783         1.70
-5.95167        -0.21898         0.76709         1.70
-6.90522         0.20352        -1.18646         1.70
-5.46910         0.19338        -0.63446         1.70
-7.59768         1.25653         0.63447         1.70
-7.11512         1.66889        -0.76708         1.70
-5.66812         1.67064        -0.21782         1.70
-6.16156         1.24639         1.18646         1.70
-8.57631         1.39044         1.14111         1.20
-7.30535         2.58276        -1.36790         1.20
-4.92868         2.47879        -0.40397         1.20
-4.49047         0.05947        -1.14110         1.20
-5.67467         1.86024         1.97288         1.20
-5.76144        -1.13285         1.36791         1.20
-7.39212        -0.41032        -1.97287         1.20
-8.13810        -1.02888         0.40398         1.20
```

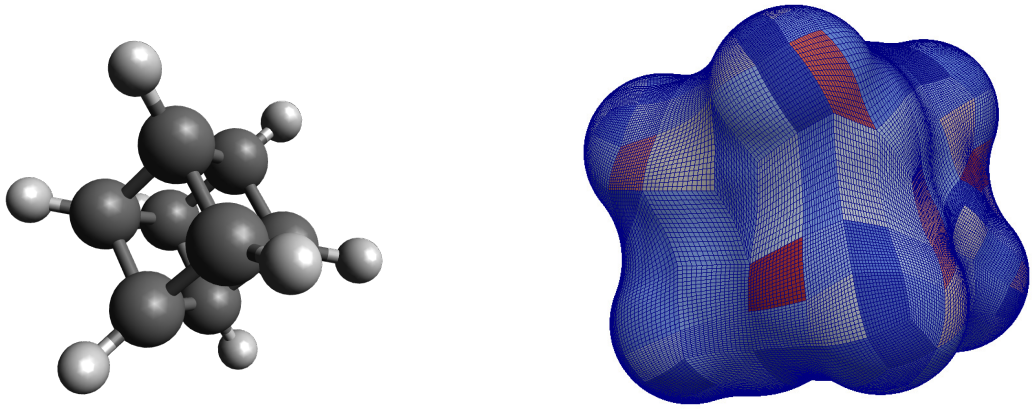


Figure A.14. Ball and stick description of cubane and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 253$, $fit = 0.401$.

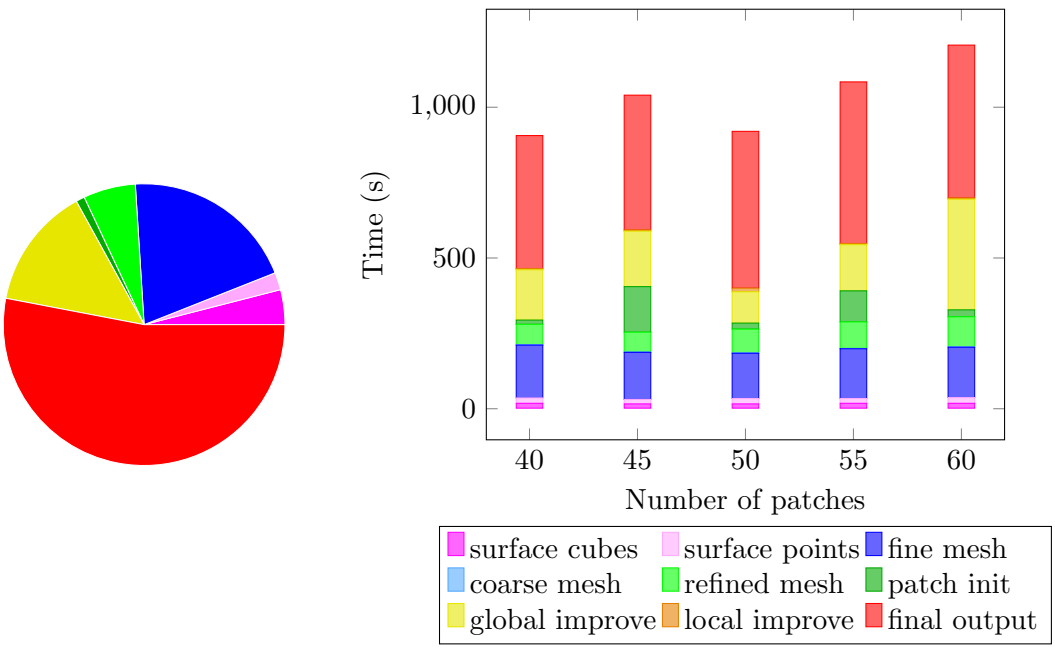


Figure A.15. Time distribution for cubane. On the left the automatic patch number detection can be seen, total time = 2760 sec. On the right the version for fixed number of patches is depicted.

Listing A.16 Input file glucose.

```
24
-0.29225        -2.79946         0.21441        1.70
 0.82402        -2.76264         0.20176        1.20
-0.80114        -1.86295         1.32573        1.70
-0.72946        -4.11698         0.42357        1.52
-0.75966        -2.30135        -1.17564        1.70
-0.39782        -0.41698         0.98412        1.70
 0.71505        -0.33081         0.97343        1.20
-0.94621         0.46866         1.92663        1.52
-0.91695        -0.06845        -0.43387        1.70
-0.37633        -0.95321        -1.40819        1.52
-0.30189        -2.24863         2.58247        1.52
-1.90886        -1.91492         1.38008        1.20
-0.54660         1.36856        -0.83176        1.70
-2.02947        -0.11090        -0.44874        1.20
-2.13038        -2.49836        -1.44352        1.52
-0.21823        -2.93076        -1.91390        1.20
-2.65971        -2.29288        -0.63458        1.20
-0.11245        -4.52357         1.08648        1.20
 0.68837        -2.18182         2.54853        1.20
-0.33193         0.47678         2.70641        1.20
 0.84578         1.53249        -0.90184        1.52
-1.00075         2.10604        -0.12955        1.20
-0.97221         1.59265        -1.83311        1.20
 1.14280         1.87741        -0.01970        1.20
```

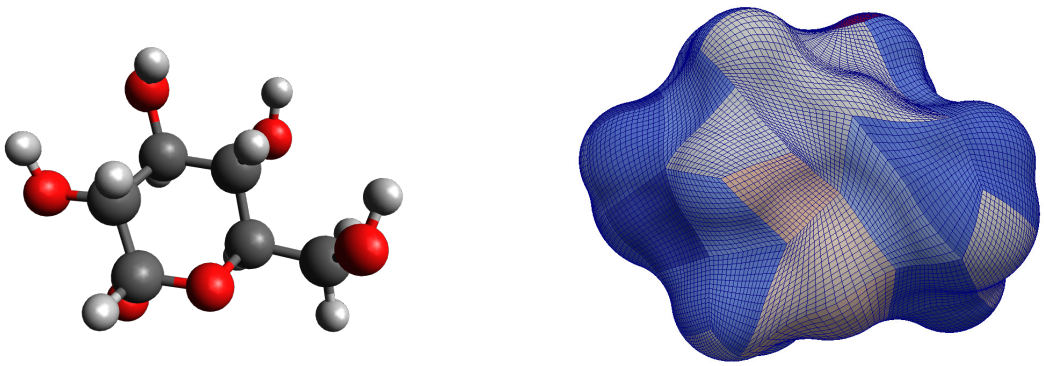


Figure A.17. Ball and stick description of glucose and the mesh resulted by automatic detection of the number of patches on level 4, $n_{\square} = 57$, $fit = 3.55$.

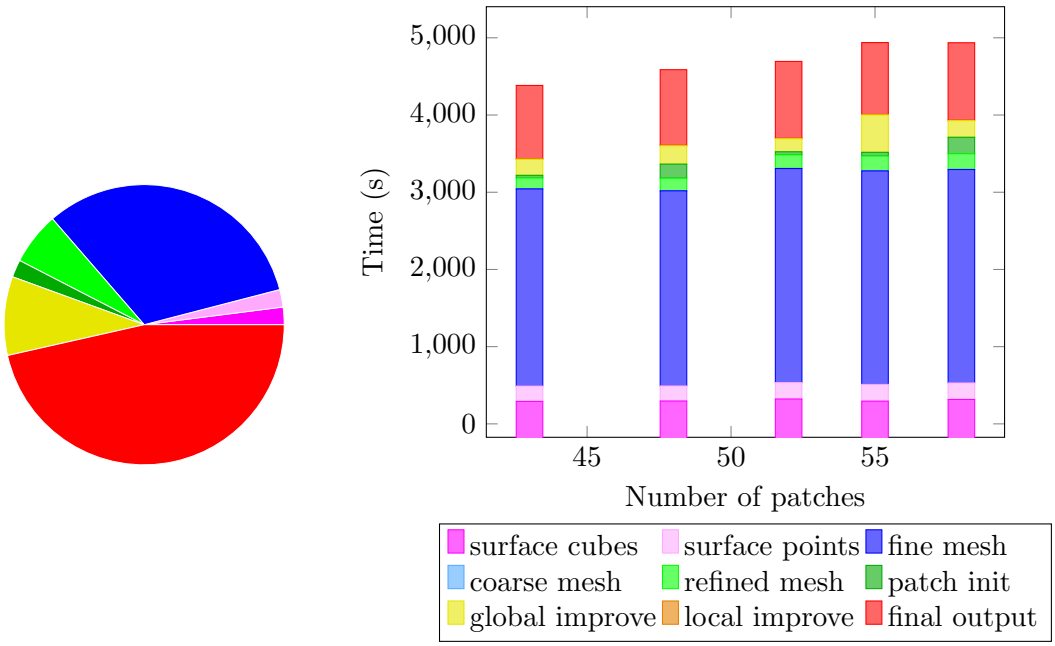


Figure A.18. Time distribution for glucose. On the left the automatic patch number detection can be seen, total time = 2567 sec. On the right the version for fixed number of patches is depicted.

Bibliography

[1] K. Aidas, C. Angeli, K. L. Bak, V. Bakken, R. Bast, L. Boman, O. Christiansen, R. Cimiraglia, S. Coriani, P. Dahle, E. K. Dalskov, U. Ekström, T. Enevoldsen, J. J. Eriksen, P. Ettenhuber, B. Fernández, L. Ferrighi, H. Fliegl, L. Frediani, K. Hald, A. Halkier, C. Hättig, H. Heiberg, T. Helgaker, A. C. Hennum, H. Hettema, E. Hjertenæs, S. Høst, I.-M. Høyvik, M. F. Iozzi, B. Jansík, H. J. Aa. Jensen, D. Jonsson, P. Jørgensen, J. Kauczor, S. Kirpekar, T. Kjærgaard, W. Klopper, S. Knecht, R. Kobayashi, H. Koch, J. Kongsted, A. Krapp, K. Kristensen, A. Ligabue, O. B. Lutnæs, J. I. Melo, K. V. Mikkelsen, R. H. Myhre, C. Neiss, C. B. Nielsen, P. Norman, J. Olsen, J. M. H. Olsen, A. Osted, M. J. Packer, F. Pawlowski, T. B. Pedersen, P. F. Provasi, S. Reine, Z. Rinkevicius, T. A. Ruden, K. Ruud, V. V. Rybkin, P. Sałek, C. C. M. Samson, A. S. de Merás, T. Saue, S. P. A. Sauer, B. Schimmelpfennig, K. Sneskov, A. H. Steindal, K. O. Sylvester-Hvid, P. R. Taylor, A. M. Teale, E. I. Tellgren, D. P. Tew, A. J. Thorvaldsen, L. Thøgersen, O. Vahtras, M. A. Watson, D. J. D. Wilson, M. Ziolkowski, and H. Ågren. The Dalton quantum chemistry program system. *WIREs Comput. Mol. Sci.*, 4(3): 269–284, 2014.

[2] M. Bebendorf. Approximation of boundary element matrices. *Num. Math.*, 86(4): 565–589, 2000.

[3] A. D. Becke. Density-functional exchange-energy approximation with correct asymptotic behavior. *Phys. Rev. A*, 38:3098–3100, 1988.

[4] A. D. Becke. Density-functional thermochemistry. III. The role of exact exchange. *J. Chem. Phys.*, 98(7):5648–5652, 1993.

[5] M. Bugeanu, R. Di Remigio, K. Mozgawa, S. S. Reine, H. Harbrecht, and L. Frediani. Wavelet formulation of the polarizable continuum model. II. Use of piecewise bilinear boundary elements. *Phys. Chem. Chem. Phys.*, 17(47):31566–31581, 2015.

[6] E. Cancès and B. Mennucci. New applications of integral equations methods for solvation continuum models: Ionic solutions and liquid crystals. *J. Math. Chem.*,

23:309–326, 1998.

[7] E. Cancès, B. Mennucci, and J. Tomasi. A new integral equation formalism for the polarizable continuum model: Theoretical background and applications to isotropic and anisotropic dielectrics. *J. Chem. Phys.*, 107:3032–3041, 1997.

[8] J. Čížek. On the correlation problem in atomic and molecular systems. Calculation of wavefunction components in Ursell-type expansion using quantum-field theoretical methods. *J. Chem. Phys.*, 45(11):4256–4266, 1966.

[9] A. Cohen, I. Daubechies, and J. Feauveau. Biorthogonal bases of compactly supported wavelets. *Pure Appl. Math.*, 45:485–560, 1992.

[10] M. L. Connolly. Analytical molecular surface calculation. *J. Appl. Crystallogr.*, 16 (5):548–558, 1983.

[11] M. Costabel. Boundary integral operators on Lipschitz domains: Elementary results. *SIAM J. on Math. Anal.*, 19(3):613–626, 1988.

[12] C. J. Cramer. *Essentials of Computational Chemistry: Theories and Models*. Wiley, West Sussex, 2002.

[13] C. Curutchet, A. Muñoz Losa, S. Monti, J. Kongsted, G. D. Scholes, and B. Mennucci. Electronic energy transfer in condensed phase studied by a polarizable QM/MM model. *J. Chem. Theory Comput.*, 5(7):1838–1848, 2009.

[14] W. Dahmen, A. Kunoth, and K. Urban. Biorthogonal spline-wavelets on the interval. Stability and moment conditions. *Appl. Comput. Harmon. Anal.*, 6:132–196, 1999.

[15] W. Dahmen, H. Harbrecht, and R. Schneider. Compression techniques for boundary integral equations. Asymptotically optimal complexity estimates. *SIAM J. Numer. Anal.*, 43(6):2251–2271, 2006.

[16] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, 1959.

[17] P. A. Dirac. Quantum mechanics of many-electron systems. *Proc. Math. Phys. Eng. Sci.*, 123(792):714–733, 1929.

[18] V. Fock. Näherungsmethode zur Lösung des quantenmechanischen Mehrkörperproblems. *Z. Phys.*, 61(1):126–148, 1930.

[19] J. B. Foresman, T. A. Keith, K. B. Wiberg, J. Snoonian, and M. J. Frisch. Solvent effects. 5. Influence of cavity shape, truncation of electrostatics, and electron correlation on ab initio reaction field calculations. *J. Phys. Chem.*, 100(40):16098–16104, 1996.

[20] L. Greengard. The rapid evaluation of potential fields in particle systems, 1988.

[21] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325 – 348, 1987.

[22] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the laplace equation in three dimensions. *Acta numerica*, 6:229–269, 1997.

[23] W. Hackbusch. *Integral Equations. Theory and Numerical Treatment.* Birkhäuser, Basel, 1995.

[24] W. Hackbusch. *Hierarchische Matrizen: Algorithmen und Analysis.* Springer, Dordrecht, 2009.

[25] W. Hackbusch and Z. P. Nowak. On the fast matrix multiplication in the boundary element method by panel clustering. *Num. Math.*, 54(4):463–491, 1989.

[26] H. Harbrecht. *Wavelet Galerkin schemes for the boundary element method in three dimensions.* PhD thesis, Technische Universität Chemnitz, Germany, 2001.

[27] H. Harbrecht and M. Peters. Comparison of fast boundary element methods on parametric surfaces. *Comput. Methods Appl. Mech. Eng.*, 261-262:39–55, 2013.

[28] H. Harbrecht and M. Randrianarivony. Wavelet BEM on molecular surfaces: parametrization and implementation. *Computing*, 86:1–22, 2009.

[29] H. Harbrecht and M. Randrianarivony. Wavelet BEM on molecular surfaces: solvent excluded surfaces. *Computing*, 92:335–364, 2011.

[30] H. Harbrecht and R. Schneider. Biorthogonal wavelet bases for the boundary element method. *Math. Nachr.*, 269(1):167–188, 2004.

[31] H. Harbrecht and R. Schneider. Wavelet Galerkin schemes for boundary integral equations. Implementation and quadrature. *SIAM J. Sci. Comput.*, 27(4):1347–1370, 2006.

[32] D. Hartree. The wave mechanics of an atom with a non-Coulomb central field. Part I. Theory and methods. *Math. Proc. Cambridge Philos. Soc.*, 24(1):89–110, 1928.

[33] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. of Research of NBS*, 49(6):409–436, 1952.

[34] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864–B871, 1964.

[35] J. D. Jackson. *Classical Electrodynamics.* Wiley, New York, NY, 3rd edition, 1999.

[36] F. Jensen. *Introduction to Computational Chemistry.* Wiley, West Sussex, 2006.

[37] V. Karasiev, T. Sjostrom, J. Dufty, and S. Trickey. Local density approximation exchange-correlation free-energy functional. *Bull. Am. Phys. Soc.*, 59(1), 2014.

[38] J. G. Kirkwood. On the theory of strong electrolyte solutions. *J. Chem. Phys.*, 2 (11):767–781, 1934.

[39] J. G. Kirkwood. The dielectric polarization of polar liquids. *J. Chem. Phys.*, 7 (10):911–919, 1939.

[40] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133–A1138, 1965.

[41] V. Kolmogorov. Blossom V: a new implementation of a minimum cost perfect matching algorithm. *Math. Prog. Comp.*, 1(1):43–67, 2009.

[42] R. Kress. *Linear Integral Equations*. Springer, Berlin, 1989.

[43] A. W. Lange and J. M. Herbert. Polarizable continuum reaction-field solvation models affording smooth potential energy surfaces. *J. Phys. Chem. Lett.*, 1(2): 556–561, 2010.

[44] A. W. Lange and J. M. Herbert. A smooth, nonsingular, and faithful discretization scheme for polarizable continuum models: the switching/Gaussian approach. *J. Chem. Phys.*, 133(24):244111, 2010.

[45] B. Lee and F. Richards. The interpretation of protein structures: Estimation of static accessibility. *J. Mol. Biol.*, 55(3):379 – IN4, 1971.

[46] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[47] S. Miertuš, E. Scrocco, and J. Tomasi. Electrostatic interaction of a solute with a continuum. A direct utilizaion of ab initio molecular potentials for the prevision of solvent effects. *J. Chem. Phys.*, 55(1):117–129, 1981.

[48] J. M. Olsen, K. Aidas, and J. Kongsted. Excited states in solution through polarizable embedding. *J. Chem. Theory Comput.*, 6(12):3721–3734, 2010.

[49] L. Onsager. Electric moments of molecules in liquids. *J. Am. Chem. Soc.*, 58(8): 1486–1493, 1936.

[50] R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, U. Bozkaya, A. Y. Sokolov, R. Di Remigio, R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill. Psi4 1.1: An open-source electronic structure program emphasizing automation, advanced libraries, and interoperability. *J. Chem. Theory Comput.*, 2017.

[51] J. L. Pascual-Ahuir and E. Silla. GEPOL: An improved description of molecular surfaces. I. Building the spherical surface set. *J. Comput. Chem.*, 11(9):1047–1060, 1990.

[52] J. L. Pascual-Ahuir, E. Silla, and I. Tuñon. GEPOL: An improved description of molecular surfaces. III. A new algorithm for the computation of a solvent-excluding surface. *J. Comput. Chem.*, 15(10):1127–1138, 1994.

[53] W. Pauli. Über den Zusammenhang des Abschlusses der Elektronengruppen im Atom mit der Komplexstruktur der Spektren. *Z. Phys.*, 31(1):765–783, 1925.

[54] C. S. Pomelli and J. Tomasi. DefPol: New procedure to build molecular surfaces and its use in continuum solvation methods. *J. Comput. Chem.*, 19(15):1758–1776, 1998.

[55] C. S. Pomelli, J. Tomasi, M. Cossi, and V. Barone. Effective generation of molecular cavities in polarizable continuum model by DefPol procedure. *J. Comput. Chem.*, 20(16):1693–1701, 1999.

[56] J. A. Pople, J. S. Binkley, and R. Seeger. Theoretical models incorporating electron

correlation. *Int. J. Quantum Chem.*, 10(S10):1–19, 1976.

[57] C. Quan and B. Stamm. Mathematical analysis and calculation of molecular surfaces. *J. Comput. Phys.*, 322:760–782, 2016.

[58] C. Quan and B. Stamm. Meshing molecular surfaces based on analytical implicit representation. *J. Mol. Graph. Model.*, 71:200–210, 2017.

[59] R. D. Remigio, R. Bast, L. Frediani, and T. Saue. Four-component relativistic calculations in solution with the Polarizable Continuum Model of solvation: Theory, implementation, and application to the group 16 dihydrides H2X (X = O, S, Se, Te, Po). *J. Phys. Chem.*, 119(21):5061–5077, 2015.

[60] M. Repisky, S. Komorovsky, V. G. Malkin, O. L. Malkina, M. Kaupp, K. Ruud, with contributions from R. Bast, U. Ekstrom, S. Knecht, O. I. Malkin, and E. Malkin. ReSpect, version 3.3.0 (beta) Relativistic Spectroscopy DFT program, 2013.

[61] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *J. of Comput. Phys.*, 60(2):187–207, 1985.

[62] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2003.

[63] S. Sauter and C. Schwab. Quadrature for *hp*-Galerkin BEM in $\mathbb{R}^3$. *Num. Math.*, 78(2):211–258, 1997.

[64] G. Scalmani and M. J. Frisch. Continuous surface charge polarizable continuum models of solvation. I. General formalism. *J. Chem. Phys.*, 132(11):114110, 2010.

[65] E. Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Phys. Rev.*, 28(6):1049–1070, 1926.

[66] H. M. Senn and W. Thiel. QM/MM methods for biomolecular systems. *Angew. Chem. Int. Edit.*, 48(7):1198–1229, 2009.

[67] E. Silla, I. Tuñón, and J. L. Pascual-Ahuir. GEPOL: An improved description of molecular surfaces II. Computing the molecular area and volume. *J. Comput. Chem.*, 12(9):1077–1088, 1991.

[68] J. C. Slater. Note on Hartree's method. *Phys. Rev.*, 35:210–211, 1930.

[69] O. Steinbach. *Numerical Approximation Methods for Elliptic Boundary Value Problems. Finite and Boundary Elements.* Springer, New York, 2008.

[70] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*, volume 12 of *Texts in Applied Mathematics*. Springer, New York, 2002.

[71] P. Su and H. Li. Continuous and smooth potential energy surface for conductorlike screening solvation model using fixed points with variable areas. *J. Chem. Phys.*, 130(7):074109, 2009.

[72] J. Tao, J. P. Perdew, V. N. Staroverov, and G. E. Scuseria. Climbing the density functional ladder: Nonempirical meta-generalized gradient approximation designed for molecules and solids. *Phys. Rev. Lett.*, 91:146401, 2003.

[73] J. Tomasi, B. Mennucci, and R. Cammi. Quantum mechanical continuum solvation

models. *Chem. Rev.*, 105(8):2999–3094, 2005.

[74] V. Weijo, M. Randrianarivony, H. Harbrecht, and L. Frediani. Wavelet formulation of the Polarizable Continuum Model. *J. Comput. Chem.*, 31(7):1469–1477, 2010.

[75] J. Wloka. *Partial Differential Equations*. Cambridge University Press, Cambridge, 1987.

[76] D. C. Young. *Computational Chemistry: A Practical Guide for Applying Techniques to Real World Problems*. Wiley, New York, 2002.

Curriculum Vitae

Personal Data

Name	Monica Bugeanu
Date of birth	5th of October 1985
Place of birth	Bucharest
Nationality	Romanian

Education

Oct 2012 – Sep 2017	PhD student in Mathematics *University of Basel, Basel*
Oct 2009 – Feb 2012	MSc student in Simulation Sciences *RWTH Aachen University, Aachen* *Graduation: February 2012*
Oct 2004 – Sep 2009	Diploma student in Computer Science *Politehnica University of Bucharest, Bucharest* *Graduation: September 2009*
Sept 1992 – Jun 2004	School *Deutsches Goethe Kolleg, Bucharest*