Delft University of Technology

Final Project: Part B
TW3725TU

Adaptive Octree Mesh Refinement Using an Operator Recovery Error Source Detector

Group 1

Student Numbers	Name
4869982	Lars La Heij
4881699	Rinto de Vries
4899717	Tuhin Das
4856198	Terrence Dahoe

February 3, 2021

Abstract

In many geophysical electromagnetic problems, a domain is discretized as a mesh prior to solving Maxwell's equations on this mesh using numerical methods. While a fine mesh can lead to accurate solutions, the computation times tend to increase, which leads to the idea to use meshes that combine fine and coarse cells. We show that an adaptive mesh refinement algorithm can generate a mesh that has fine cells in critical areas and coarse cells elsewhere. Our algorithm represents a domain as an octree mesh and iteratively refines the mesh in locations where the error is largest. The Operator Recovery Error Source Detector (ORESD) error estimator is used to find these locations. The algorithm succeeds to converge and generate refined meshes for multiple models. Convergence can take a long time due to multiple iterations, but the generated mesh can be efficiently used by numerical solvers due to low number of cells. However, electric field solutions generated on the refined mesh are 15.8% less accurate in critical areas than solutions generated on uniform meshes. The algorithm can thus best be used to generate meshes that can be efficiently reused multiple times to inspect large models, prior to using more expensive meshes for more accurate solutions.

Keywords: electromagnetic methods, adaptive mesh refinement, octree mesh, error estimator

Executive Summary

Electromagnetic methods in geophysical applications often require solving Maxwell's equations using numerical methods, to approximate values of an electric field (EF). For these methods, the first step is the discretization of the domain and the 3D model with a mesh. An important trade-off for a discretization is the minimum cell size, as the number of cells, which is dependent on the cell size, affects the computation time and the solution accuracy. However, using adaptive meshing it is possible to generate a mesh that has small cells in important locations where high accuracy is required, and coarse cells elsewhere. We show a concept for an adaptive meshing algorithm and compare how an adapted mesh performs against a conventional uniform grid.

Our adaptive algorithm first generates an octree mesh of a 3D model, where the model surface is refined with small cells and the other parts of the domain are left coarse. The mesh is iteratively refined in certain locations such that the solution can be improved. By utilizing the Operator Recovery Error Source Detector (ORESD) error estimator, certain locations in the mesh can be detected where the mesh should be refined.

The ORESD estimator computes the curl of the EF in two different ways and compares the different curl values at the same coordinates. First, the curl C_1 is directly computed with a numerical approximation from the existing mesh. For curl C_2, the EF is first computed with a numerical approximation and is then interpolated. The curl C_2 is then computed from this interpolated EF. By comparing C_1 and C_2 in each cell, the algorithm determines in which cells the solution is not accurate enough. Afterwards, 5% of the cells with the highest differences between C_1 and C_2 are refined in the octree mesh. In the next iteration the new refined mesh is used to generate approximations of the EF, C_1 and C_2 and they are again compared. This process is repeated until the average difference between the solutions of the EF in the cells drops below a certain threshold.

We ran multiple experiments to investigate various qualities of the adaptive algorithm. In the first experiment we tested whether the algorithm converges for different 3D models. Convergence is achieved when the difference of the EF solutions between consecutive iterations drops below a given threshold. We ran the algorithm on a basic block, a rotated block and a perfect sphere model with a threshold of 0.01 and determined that the algorithm indeed converges for every model. In the second experiment we investigated the convergence of the algorithm for a larger realistic model. The algorithm succeeded to converge for this model too.

In the third experiment we compared the actual EF solution on a refined octree mesh with a benchmark solution computed on a tensor mesh. The benchmark solution was generated with `SimPEG` using a cell width of 50. The solution on the refined octree mesh was 15.8% less accurate around the receivers, but the number of cells in the octree mesh was 93 times lower.

In the fourth experiment we investigated the algorithm computation time as a function of the number of mesh cells and compared this with `emg3d`. It showed that the computation time is exponentially dependent on the number of mesh cells, while the time complexity of `emg3d` is linearly dependent on the number of cells and thus is more efficient.

Based on the results we saw that the adaptive algorithm manages to converge and generate a refined mesh for a given model with acceptable accuracy for the amount of cells it uses. Due to the larger computation times, the adaptive algorithm mainly shows potential in generating custom meshes for models that can be reused multiple times, which can eventually save computation time. For future studies we recommend investigating a minimum required level of refinement in the octree mesh and exploring the use of an error threshold as cell refinement condition in order to save computation time. The parallelization of the error estimator can possibly lead to optimized performance.

Table of Contents

1 Introduction

Electromagnetic methods (EM) are widely used in geophysics for a broad variety of applications that range from exploration of natural resources to the measuring of glacier thickness [1]. In these methods, the interactions of an electromagnetic field with the environment are observed and analyzed. These analyses typically involve solving Maxwell's equations, which are used to describe the behaviour of electromagnetic fields.

The solutions of Maxwell's equations are often approximated using numerical methods such as the Finite Element Method [2], Finite Difference Method [3], Finite-Volume Method [4] or Finite-Integration Method [5]. These numerical methods discretize a 3D model and the domain before approximating the electric field on the discretized grid. The method of domain discretization can highly influence the accuracy of a numerical solver [6]. Thus, it is important to use an adequate domain discretization for a given numerical problem.

An important discretization characteristic is the grid granularity, which defines how small the mesh cells are. The mesh should be as fine as possible to properly model the environment and the interactions [7], but should also be coarse enough to save computational costs. To benefit from both worlds, one option would be to generate a mesh that is fine in certain areas and coarse in other areas. Such a mesh can also adaptively be refined in certain areas to iteratively improve the numerical solution [8].

The questions we ask in this study are how we can develop such an adaptive algorithm and how the adapted mesh compares to a uniform mesh grid. We are interested in the difference between computation times and the solution accuracy of the different mesh types. The solvers we will use for this comparison are `SimPEG` and `emg3d`. These are both open source solvers for 3D electromagnetic models that use uniform grids [9, 10]. `SimPEG` also supports solving on non-uniform grids and therefore this solver was also integrated into the adaptive algorithm.

For the adaptive algorithm, we present an algorithm that can automatically generate a mesh for a given problem and can adaptively refine the mesh in certain areas. We describe how the algorithm generates a base mesh for a 3D model using an octree discretization technique [11], leading to a mesh that is refined around the model and coarse elsewhere. Further, we show how the algorithm uses an error estimator [12] to determine locations in the grid that should be further refined. The software iteratively refines the grid and compares the new solution with the previous solution to find new locations to refine the mesh, resulting in an iteratively more accurate solution. Using various experiments we plan to compare this adaptive algorithm with `emg3d` regarding complexity and with `SimPEG` regarding solution accuracy.

This document is organised in the following structure. Section 2 presents relevant results from our literature study, based on which the adaptive algorithm has been developed. In section 3 we describe in depth how our adaptive algorithm generates meshes and adaptively refines meshes to create more accurate numerical solutions. Section 4 contains the experiments we have performed to analyse the convergence, accuracy and scalability of the adaptive algorithm. We discuss the algorithm and give future recommendations in section 5, after which we summarize and conclude our findings in section 6.

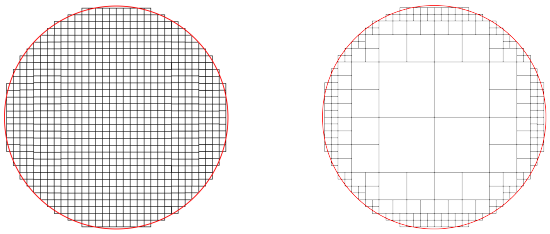
2 Literature Review

Prior to this study we conducted a literature review concerning meshing methods and discretization errors. The conclusion of the literature review was that there is still a knowledge gap regarding the optimal method for iterating and adapting models in contemporary codes and solvers. In this chapter we outline the points from the literature review that serve as a basis for the final adaptive algorithm.

First we discuss the Modified-Octree Method in section 2.1, based on which we chose to use octree meshes. Section 2.2 presents a phenomenon called the staircase effect, which is error induced by not properly discretizing curved or diagonal faces and edges. Finally, section 2.3 discusses the a posteriori error estimator that is used to detect locations in a mesh where the errors are the largest.

2.1 Modified-Octree Technique

The method for 3D mesh generation called the Modified-Octree Technique is described in [11]. A similar method for 2D is called the Modified-Quadtree Technique and is presented here. For 2D quadtree mesh generation, the domain is first encoded in a quadtree data-structure as follows. First, a square is created that surrounds the domain. The square is divided into four quadrants, each of which is tested whether it is inside the domain (full), outside the domain (empty) or partially inside the domain (partial). Full and empty quadrants are left untouched and partial quadrants are subdivided again into four quadrants and the test process is repeated. As this procedure is recursive in nature, the structure and succession of these quadrants can be stored in a tree-like structure such as a quadtree. The partial quadrants are subdivided until the desired tree-depth is achieved. Quadtree meshes built using these rules usually lead to a lower number of mesh cells. Fig. 1 shows how a circular domain is discretized into a uniform mesh and a quadtree mesh. It can be clearly seen that the uniform mesh contains much more cells than the quadtree mesh.



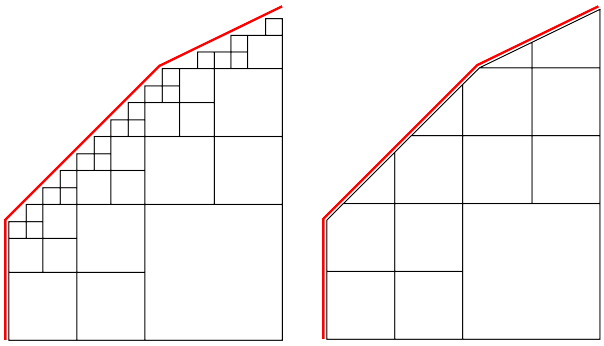
(a) Uniform mesh. (b) Quadtree mesh.

Figure 1: Two possible mesh representations of a circle shaped domain, which is shown in red. On the left a uniform mesh and on the right a quadtree mesh. Figures based on [11].

Quadtree meshes created with this method are usually not fit for numerical methods yet, as the aspect ratios between neighbouring cells can be large and the complete mesh can lack smoothness. By adding certain constraints during the quadtree creation and with some additional post-processing, it is possible to turn the domain subdivision into a suitable mesh.

The first constraint is that neighbouring quadrants can differ in only one level of subdivision, which is not necessary in a regular quadtree structure. The second constraint states that quad-

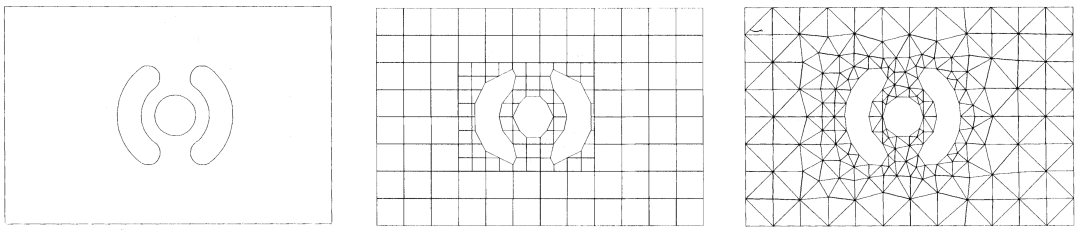
rants at the domain boundaries can have cut corners to adapt to the boundary. The result of this constraint is that the quadtree does not need small fine elements to approximate edges or curves. Instead, larger elements can be cut and used to approximate edges more accurately, as shown in Fig. 2.



(a) Without cut edges. (b) With cut edges.

Figure 2: Part of a domain with slanted edges is shown in red, which is to be approximated using a quadtree structure. A regular quadtree subdivision is shown in (a) and it is clear that due to a staircase effect, the squares can not approximate the borders. An improved quadtree is shown in (b), where cut edges are allowed. The mesh in (b) does not suffer from the same staircase effect.

After creating a mesh using these rules, the mesh is further refined using various rules. First each quadrilateral in the mesh is split up into smaller triangles or quadrilaterals. This step is done to improve the smoothness of the grid, however, at the cost of the skewness and aspect ratios of the elements. Afterwards, the resulting inner nodes of the mesh are moved towards the centroids of their neighboring nodes, which again improves the skewness and aspect ratios of the elements, leading to a final unstructured triangular mesh. The complete process of 2D mesh generation using this technique is displayed in Fig. 3.



(a) The original domain. (b) Quadtree domain representation. (c) The final unstructured triangular mesh.

Figure 3: Creating a 2D mesh from a domain using the Modified-Quadtree Technique. The original domain as shown in (a) is turned into a quadtree structure using additional constraints, leading to the structure in (b). Finally the resulting quadrilaterals are subdivided into triangles and the internal points are shifted, resulting in the final 2D mesh in (c) [11].

This technique can be adapted to 3D by using an octree structure. In this case the original domain is surrounded by a cube, and the cube octants are recursively split into finer cubes. By

adding more constraints and extra steps, this method leads to the Modified-Octree Technique that can be used to generate 3D meshes of domains.

2.2 Staircase Effect in Meshes

The staircase effect becomes present when a shape in the domain is not aligned with a regular grid. The geometry of the shape can not be fully represented anymore because of the nature of the grid. The result of the staircase effect is that the approximated solutions at the shape boundaries are sensitive to errors. An illustration of the staircase effect is shown in Fig. 4.

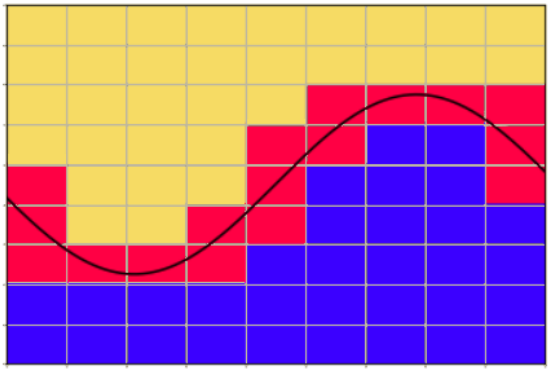


Figure 4: A visualisation of the staircase effect. A border in the shape of a sine function is shown between two parts of the domain. It is not possible to approximate the smooth sine border accurately on a coarse Cartesian grid.

Fig. 4 shows a domain that is split in an upper domain and a lower domain by a sine function. The lower and upper domain each represent a different medium. The to be represented boundary is the black line. The boundary is approximated by the red cells and it is clear that the discretization does not align well to the shape of the boundary. The staircase effect is thus prevalent. As a result of the coarse grid the approximation of a smooth line on a Cartesian grid is not accurate.

In Fig. 5 the same method is applied, but the grid is finer than in Fig. 4. The result is that the staircase effect is also less prevalent, but it is still visible that the approximation of the boundary consists of squares, where a smooth line should be present. In the right figure, the mesh is even finer and the approximation of the boundary comes closer to the smooth sine function. The finer the grid the better the approximations of smooth boundaries, which therefore leads to a smaller discretization error. It is however not economical to apply the fine mesh to the whole domain. Thus it would make sense to apply an adaptive mesh refinement scheme that mostly refines a mesh around the surface of a model and keeps other locations more coarse.

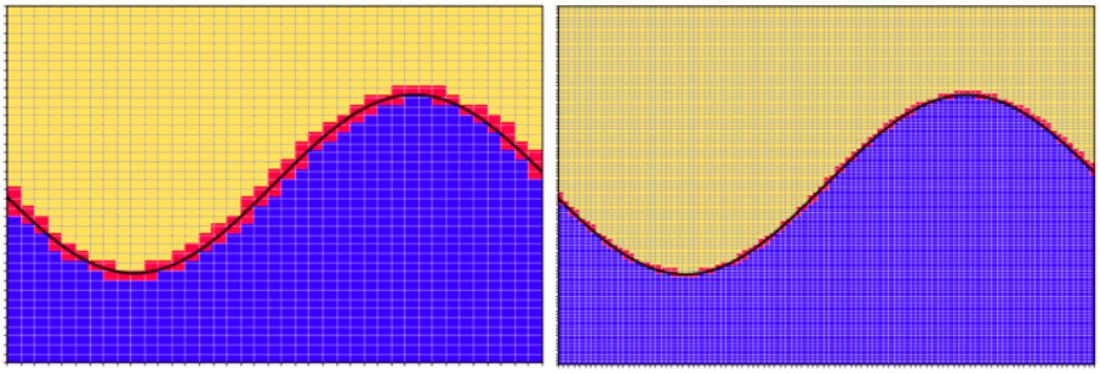


Figure 5: Two more refined versions of the mesh in Fig. 4 are shown. The fine grids allow a better approximation of the smooth sine function on a Cartesian grid, due to a less prevalent staircase effect.

2.3 Operator Recovery Error Source Detector (ORESD)

The definition of an adaptive grid is that the grid cell width and/or time-step varies throughout the domain and simulation. In this particular application, the solution is constructed in the frequency domain and only for one frequency at a time. Thus we only focused on varying the cell width in the domain. At certain locations where the solution varies rapidly, say a boundary surface where material parameters change drastically, the grid is required to be relatively fine. At other locations in the domain, the solution remains relatively constant. Therefore, the grid can be quite coarse. We do not always know exactly where the solution fluctuates a lot and where it does not. In these cases, either an error estimator or an error indicator can be used to point out these places. A large error means that the solution does fluctuate heavily at that location in the domain. In the literature review, we looked at an error estimator which is computationally quite cheap and can be implemented in a wide range of applications. This error estimator falls under the category of gradient recovery error estimators.

The method is called the Operator Recovery Error Source Detector (ORESD). It is based on discretizing an operator and on interpolation to estimate the local truncation error. The divergence and gradient operator were explored in a research conducted by Giovanni Lapenta, who developed the method [12]. An explanation will be provided in the one-dimensional case, using simple discretizations. Consider Fig. 6.

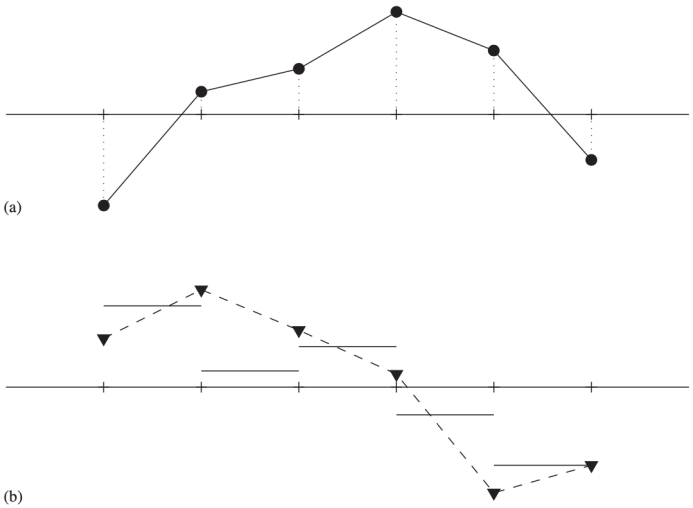


Figure 6: Example of 1-D gradient ORESD using linear interpolation [12].

In the figure two plots are given. In subplot (a) the discrete solution u is given by the solid dots. Linear interpolation was used to approximate the continuous solution. Higher interpolation might be required in other applications. In subplot (b), the triangles represent the discretized operator of $\mathrm{d}u/\mathrm{d}x$, using a central difference scheme in this example.

$$\left.\frac{\mathrm{d}u}{\mathrm{d}x}\right|_n = \frac{u_{n+1} - u_{n-1}}{2\mathrm{d}x}$$

Again, linear interpolation was used to approximate a function between the triangles, resulting in the dashed line. The solid lines present the exact first-derivative of the interpolated function $u(x)$. The error between the dashed line and the solid lines in subplot (b) is denoted as

$$e = \tilde{O}(\mathbf{x}) - \mathbf{O}\tilde{q}(\mathbf{x}), \tag{1}$$

where $\tilde{O}(\mathbf{x})$ is the exact operator of the solution (in this case exact gradient of interpolated $u(x)$) and $\mathbf{O}\tilde{q}(\mathbf{x})$ is the interpolated discretized operator (in this case the central difference scheme for $u(x)$). The error can be integrated over a cell volume using a L_p norm such as L_1, L_2 or L_∞.

$$e_c = \left(\frac{1}{V_c}\int_{V_c} \mathrm{e}^p \, \mathrm{d}V\right)^{1/p} \tag{2}$$

This error estimate was proven to be an accurate estimator for indicating locations in the domain where the grid has to be refined [12]. Bank and Xu proved that the recovered gradient is a superconvergent approximation to the gradient of the true solution as the element volume size decreases [13].

3 Methods

The previous section contains the preliminaries on which our method is built. The objective is to build a solver that uses an adaptive grid. In this section the methods are explained we use to build the adaptive algorithm. We took advantage of already existing libraries that are used for computational electromagnetic problems. Here we discuss the functionality of these libraries.

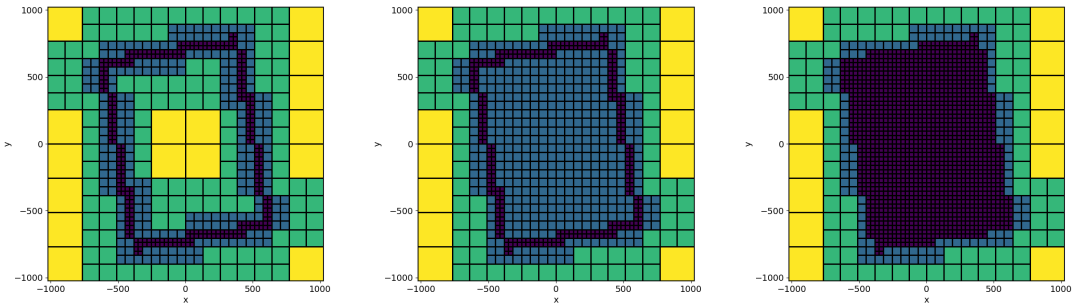
Building an adaptive meshing algorithm starts with the generation of an octree mesh. The first section discusses how the octree mesh is generated with the `discretize` library. Subsequently the equations which result in the electric field and magnetic flux density are solved on the octree method. We describe in the second section the use of the `SimPEG` library to solve these equations numerically. The last section discusses how the error estimator works and gives the iterative scheme for the refinement and the pseudocode of our adaptive meshing algorithm.

3.1 Mesh Generation With an Octree Data-Structure

The first step of the adaptive algorithm is to represent the problem model and domain in a 3D mesh. Based on the literature study we chose to represent the model with an octree data-structure, as the recursive nature of an octree gives the opportunity to refine a model at specific locations. To generate an octree mesh of a given 3D model we used the `discretize` Python library, as it works seamlessly with the `SimPEG` numerical solving library.

When representing a 3D model with a mesh it is important that only the surface is refined and the interior and exterior are as course as possible. This is because the solution usually changes most around the surface of a model and the solution changes less further away from the surface. To save computational power it is thus possible to make the mesh fine around the surface of a model, and course everywhere else. Again an octree structure is suitable to generate such a mesh.

We take a rotated brick shaped model as an example. Fig. 7 shows a few slices of the mesh we generated from such as model in the domain $[-1024, 1024]^3$. For this mesh we took 32 as the width of the smallest most refined cells. It can be seen that the octree mesh from Fig. 7 is course on the inside and the outside of the brick, and that the mesh becomes more refined near the surface.



(a) Octree mesh slice at z = 400. (b) Octree mesh slice at z = 720. (c) Octree mesh slice at z = 784.

Figure 7: Plan views of the rotated block octree mesh (7a) inside the block at $z = 400$, (7b) inside near the boundary of the block at $z = 720$, (7c) at the boundary of the block at $z = 784$.

A mesh as shown in Fig. 7 is the starting point of our adaptive mesh refinement algorithm. After creating such a base mesh, it is necessary to refine the octree mesh around the transmitters and the receivers in the given geophysical model. Transmitters and receivers are modeled as points and the meshes are refined around these points. `discretize` supports such point refinements and Fig. 8 shows how a single point would be refined by `discretize` in the previously shown mesh.

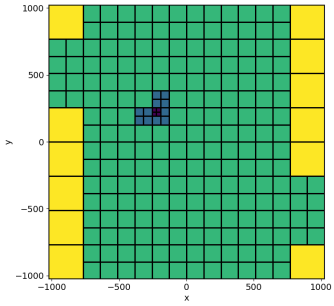


Figure 8: The refinement of a single point transmitter or receiver at $(-250, 250, 1000)$.

3.2 Mesh Error Estimation using ORESD

The final model that is constructed should represent the real-life scenario as accurately as possible. A transmitter sends an electromagnetic signal into the ground, which triggers a response. An electromagnetic field is created which is measured by the receivers on the surface. A model is accurate if the simulated data matches the measured data, with a certain tolerance. Electromagnetic phenomena are governed by Maxwell's equations. In the frequency domain they are given by:

$$\nabla \times \vec{E} + i\omega\vec{B} = \vec{S_m}, \tag{3}$$

$$\nabla \times \vec{H} - \vec{J} - i\omega\vec{D} = \vec{S_e}, \tag{4}$$

$$\nabla \cdot \vec{B} = 0, \tag{5}$$

$$\nabla \cdot \vec{D} = \rho_f, \tag{6}$$

with $\vec{E}$: electric field (V/m),
$\vec{H}$: magnetic field (A/m),
$\vec{B}$: magnetic flux density $\left(\mathrm{Wb/m^2}\right)$,
$\vec{D}$: electric displacement / electric flux density $\left(\mathrm{C/m^2}\right)$,
$\vec{J}$: electric current density $\left(\mathrm{A/m^2}\right)$,
$\vec{S_m}$: magnetic source term $\left(\mathrm{V/m^2}\right)$,
$\vec{S_e}$: electric source term $\left(\mathrm{A/m^2}\right)$,
ρ_f : free charge density $(\mathrm{C/m^3})$.

For this particular application we are only interested in the first two equations, which result in the electric field and magnetic flux density everywhere in the domain when solved. This application

uses a low-frequency range, for which the electric displacement is negligible. This leads to the quasi-static approximation:

$$\begin{aligned} \nabla \times \vec{E} + i\omega \vec{B} &= \vec{S_m}, \\ \nabla \times \vec{H} - \vec{J} &= \vec{S_e}, \end{aligned} \tag{7}$$

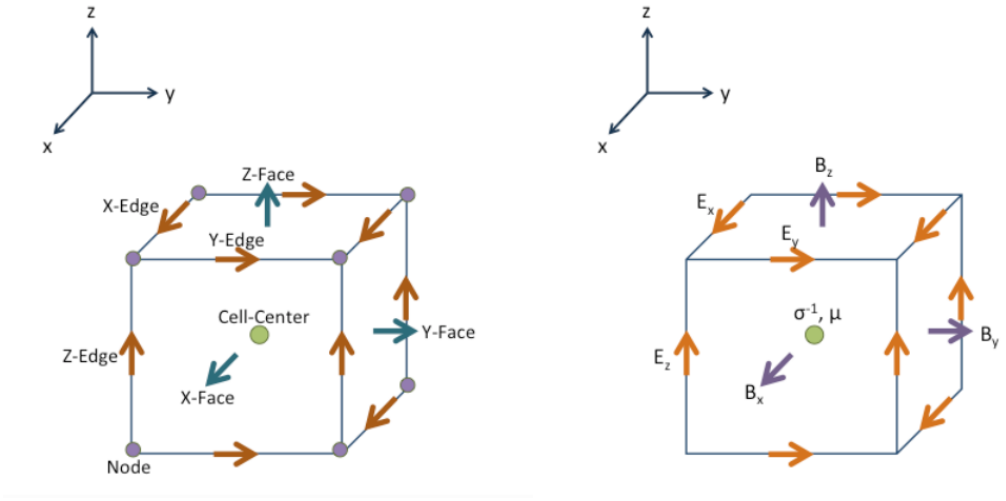
or

$$\begin{aligned} \nabla \times \vec{E} + i\omega \vec{B} &= \vec{S}_m, \\ \nabla \times \mu^{-1} \vec{B} - \sigma \vec{E} &= \vec{S}_e, \end{aligned} \tag{8}$$

since

$$\begin{aligned} \vec{J} &= \sigma \vec{E}, \\ \vec{B} &= \mu \vec{H}, \end{aligned}$$

where σ is the electrical conductivity (S/m) and μ the magnetic permeability (H/m). The `SimPEG` module solves these equations using a finite-volume discretization [4]. The electric field is solved on the cell edges, while the curl of the electric field, the magnetic flux density and source terms are solved on the faces of the cell. Material parameters such as conductivity, resistivity and magnetic permeability are defined at the cell-centers. This can be seen in Fig. 9.



(a) Finite-volume cell elements.

(b) Solution vectors and parameters.

Figure 9: The correspondence between (9a) the elements of a finite-volume cell and (9b) the solution vectors and parameters at various locations in the cell.

The ORESD method is applied in the following manner. The operator used is the curl $\nabla\times$, specifically the curl of the electric field $\nabla \times \vec{E}$. The `SimPEG` solver solves for the electric field in the domain. This electric field is interpolated to get a continuous function. E_x, E_y and E_z have to be interpolated separately using the x-, y- and z-edges from the grid respectively. After this, the components are brought together into a single vector, resulting in a continuous electric vector field. From this interpolated electric field, the curl is computed. In practice this curl operator is not completely exact. A central difference scheme with order $\mathcal{O}(h^2)$ is used to compute the derivatives. The second method used to compute the curl, called the discretized operator, is done by rewriting the first Maxwell equation.

$$\nabla \times \vec{E} = \vec{S_m} - i\omega \vec{B} \tag{9}$$

SimPEG solves for the magnetic flux density and source field. Using the equation above, we can compute the curl of the electric field. This curl is evaluated at the faces of the cell. $(\nabla \times \vec{E})_x$, $(\nabla \times \vec{E})_y$ and $(\nabla \times \vec{E})_z$ have to be interpolated separately using the x-, y- and z-faces from the grid respectively. Again, these components are brought together to get the curl field. The difference between these curls, computed in different ways, served as the error indicator.

$$e = |\nabla \times \vec{E}^1 - \nabla \times \vec{E}^2| \tag{10}$$

SimPEG uses second order accuracy to solve for the electric field and magnetic flux density. By applying the curl operator (also with second order accuracy) to the interpolated electric field, it results in a first order accurate approximation of the curl of the electric field, so $\nabla \times \vec{E}^1$ is first order accurate. $\nabla \times \vec{E}^2$ was computed only using the magnetic flux density and the magnetic source field, thus this is a second order accurate approximation. The closer the discretized field gets to the exact solution, the smaller the difference between the first order- and second order approximations will become. For the sake of computational cost, the error is integrated using the L_2 norm with a single-point approximation in the cell-center. Three different interpolation methods are implemented, namely multiquadric radial basis interpolation, linear- and nearest neighbour interpolation. The estimator supports both conductivity and resistivity models.

3.3 Adaptive Mesh Refinement with an Iterative Scheme

The error indicator plays a vital role to make the grid more automatic and adaptive. It shows where the largest errors of the numerical solution are. When the locations of the largest errors are indicated, one knows where to refine the grid. Therefore, this error estimator is used iteratively to make the model more and more accurate. After each iteration, the error estimator is used to determine the error in every cell-center of the grid. A user-defined percentage of the grid with the largest errors are flagged for refinement.

Another method is to flag all cells for which the error rises above a a certain threshold. However, as the specific error estimator had never been used before, we were not certain of a concise value for the threshold.

After the flagged cells are refined, the model moves on to the next iteration. The model stops iterating on itself once it has converged. To test for this convergence, the electric field of the previous iteration is compared to the electric field of the current iteration using Equation 11. Once the relative difference in every cell-center of the current iteration falls beneath 1%, the model has converged. This means that the mesh is at its most efficient. Further refining the mesh would only increase the cost of the mesh without improving the solution significantly in return.

$$\left\| \frac{\vec{E}_i - \vec{E}_{i-1}}{\vec{E}_{i-1}} \right\| \tag{11}$$

A summary of the iterative scheme has been depicted in Fig. 10. Algorithm 1 shows the pseudocode for the adaptive refinement scheme, based on which we made our implementation.

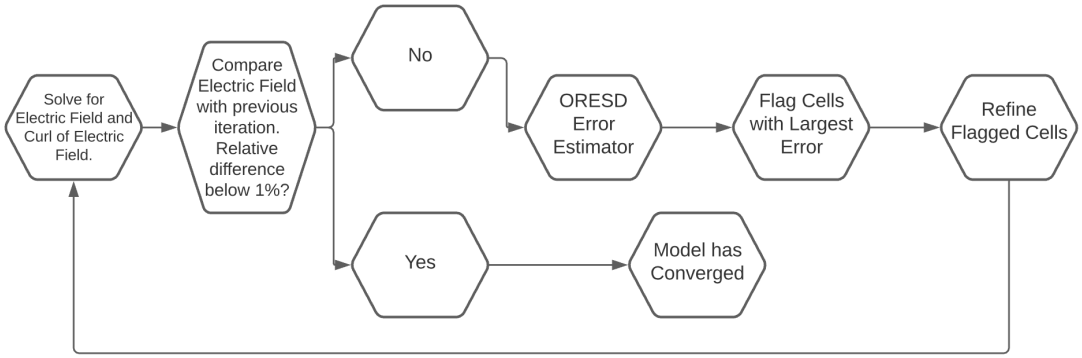


Figure 10: Iterative scheme used to automate and adapt the model.

Algorithm 1 Adaptive Mesh Refinement

Input:
D: domain
S: list of model surface coordinates
T: list of transmitter coordinates
R: list of receiver coordinates
Δx: minimum mesh cellwidth
$maxiter$: maximum number of iterations

$M \leftarrow$ octree mesh of D using Δx
refine M at locations S
refine M at locations T
refine M at locations R

$\varepsilon \leftarrow \infty$ ▷ average difference between consecutive iterations
$i \leftarrow 0$

while $\varepsilon > 0.01$ and $i < maxiter$ **do**
    $\vec{E} \leftarrow$ numerical solution of electric field
    $C_1 \leftarrow$ numerical solution of curl $\nabla \times \vec{E}$
    $C_{1,int} \leftarrow$ interpolated C_1
    $\vec{E}_{int} \leftarrow$ interpolated $\vec{E}$
    $C_{2,int} \leftarrow$ curl of $\vec{E}_{int}$

    $diff \leftarrow \|C_{2,int} - C_{1,int}\|_2$ for each cell center
    refine M at 5% of the cell centers in $diff$ with the largest value

    if $i > 0$ **then**
        $n \leftarrow$ number of cells in current M
        $\varepsilon = \frac{1}{n} \sum \|\vec{E}^i_{int} - \vec{E}^{i-1}_{int}\|_2$ ▷ sum over all cells in M
    $i = i + 1$

return $\vec{E}_{int}$, M

4 Experiments

In the previous section we described how our algorithm can generate a numerical solution for Maxwell's equations using an adaptive refinement scheme. To determine whether the software can be adequately used in the geophysical field, we conducted several experiments to test the performance of the tool and we compared the algorithm with the conventional solver `emg3d`. We conducted four experiments in total to investigate various aspects of the adaptive meshing algorithm.

The first experiment investigates the rate of convergence of the software for basic 3D models (section 4.1). The second experiment investigates the rate of convergence of the adaptive algorithm when applied to a large realistic model (section 4.2). The third experiment compares the converged octree mesh solution with the `emg3d` mesh solution using a `SimPEG` benchmark solution (section 4.3). The final experiment looks into the scalability of the algorithm as the number of cells increases and compares this with the scalability of `emg3d` (section 4.4).

4.1 Convergence of the Adaptive Algorithm on Three Basic 3D Models

In the first experiment, the convergence of the adaptive meshing algorithm was investigated with the use of basic 3D models. Each 3D shape was placed in the same domain $[-500, 4500] \times [-2000, 2000] \times [-2500, 200]$ meter. The transmitter we used was a magnetic source placed at the origin $(0, 0, 0)$. The transmitter has a frequency of 1 Hz and generates a magnetic field which leads to an electric field in the domain. The basic 3D shapes we used as models are a block, a rotated block and a sphere. The block model has been reused from an existing geophysical EM example. The model of the sphere has been created manually for the purpose of this test.

The basic 3D shapes were put into a domain where the background has a different resistivity and conductivity value. The resistivity value of the 3D shapes is 100 Ωm, while the background has a resistivity of 0.1 Ωm. Due to this difference in resistivity, the values of the electric field change drastically on the boundaries between the shape and the background. Due to the setup of our octree meshes, the grid had initial refinements at the surface of each shape such that the change of the electric field could be approximated accurately enough. The transmitters and receivers in the domain were also locally refined before the algorithm was run.

The objective of the experiment was to show that the solutions converge over the iterations. Convergence is achieved when the average relative difference of the solution in every cell-center between two consecutive iterations is less than 1%. To determine the convergence of the adaptive algorithm, we analysed the average relative difference of each iteration, for each model. We visualize the convergence by plotting the average relative difference against the iteration number. Fig. 11, 12 and 13 respectively show the convergence plots for the block, rotated block and sphere models. For the basic block, the electric field is also plotted in Fig. 17 up until 19 to give an idea of the solution $\vec{E}$ and to see whether the results make sense. The electric field of the other basic shapes is not shown in this report for the sake of repetitiveness. However, in section 4.2 the field of the large artificial model is displayed in order to again test the logicality of the results.

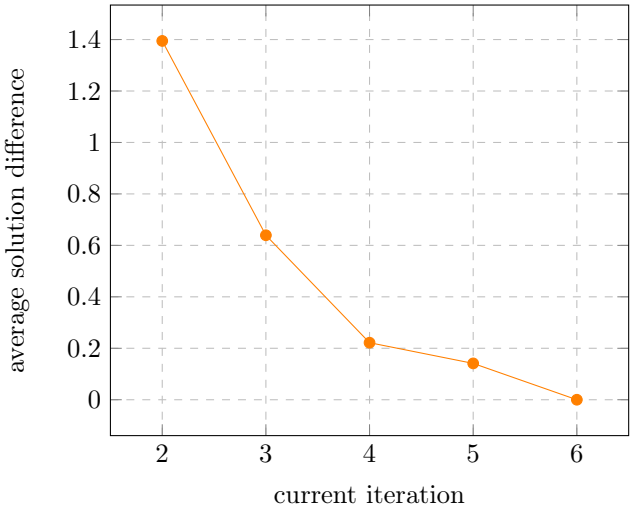


Figure 11: The convergence of the adaptive a for a simple block model. The average difference between the solutions of iteration i and $i-1$ is plotted against the iteration number i.

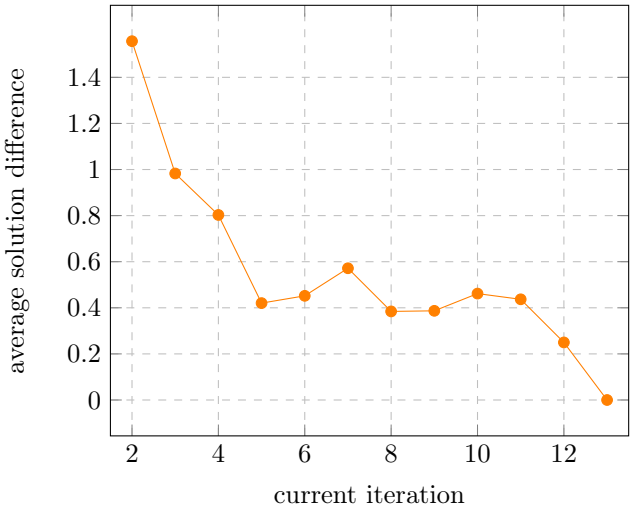


Figure 12: The convergence of the adaptive algorithm for a block model that is rotated by 15 degrees around the z-axis. The average difference between the solutions of iteration i and $i-1$ is plotted against the iteration number i.

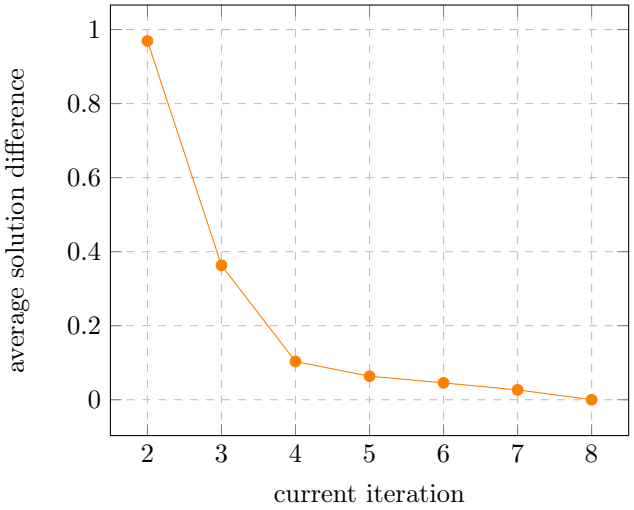


Figure 13: The convergence of the adaptive algorithm for a perfect sphere model. The average difference between the solutions of iteration i and $i-1$ is plotted against the iteration number i.

From Fig. 11, 12 and 13 we can see that the adaptive algorithm converged for every model, as the final solution difference fell below 0.01 for each model. The necessary number of iterations to reach convergence does differ between the model. Logically, the basic model has the fastest convergence within 6 iterations. It is interesting to see that the sphere model converged faster than the rotated block model, with 8 iterations versus 13 iterations respectively. Both models suffer from the staircase effect because of curved or skewed faces in the models. However, the rotated cube probably suffers more from the staircase effect than the sphere as no parts of the rotated cube are aligned with the octree mesh. The sphere at least has six sides that are aligned with the octree mesh, if we imagine a cube around the sphere. These six sides of the sphere can be modeled more closely by refining those areas in the mesh.

First we look at the results of the basic block model. The block has the shape $[200, 600] \times [-500, 500] \times [-1000, -800]$. We generated an octree mesh with minimum cell width 50 and ran the adaptive algorithm on the mesh. We take a look at the final mesh of the basic block model after convergence. Fig. 14, 15 and 16 shows multiple slices of the final mesh.

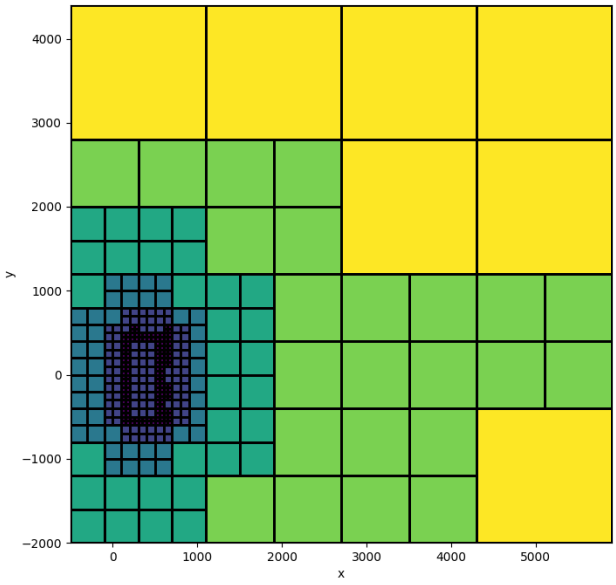


Figure 14: 2D slice at $z = -900$ of the converged octree mesh for a basic block model.

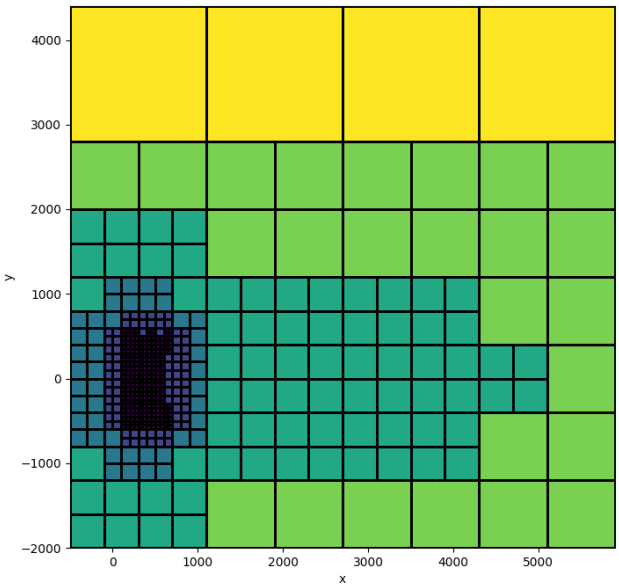


Figure 15: 2D slice at $z = -800$ of the converged octree mesh for a basic block model. This slice shows the top face of the block.

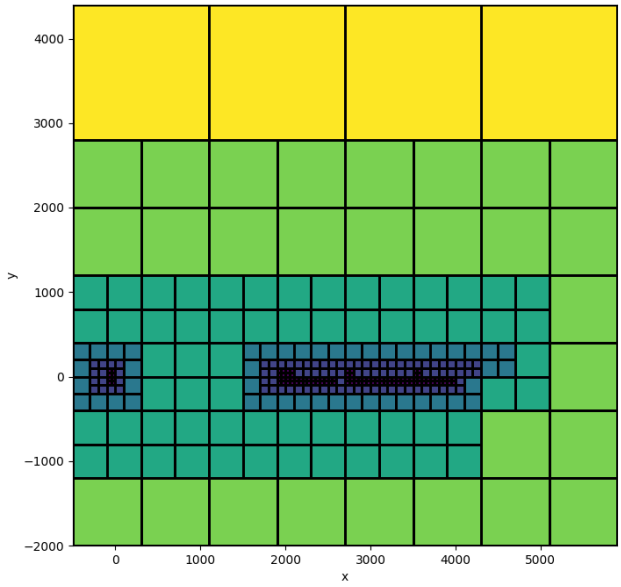


Figure 16: 2D slice at $z = -25$ of the converged octree mesh for a basic block model. On the left the transmitter is visible and on the right a line of receivers.

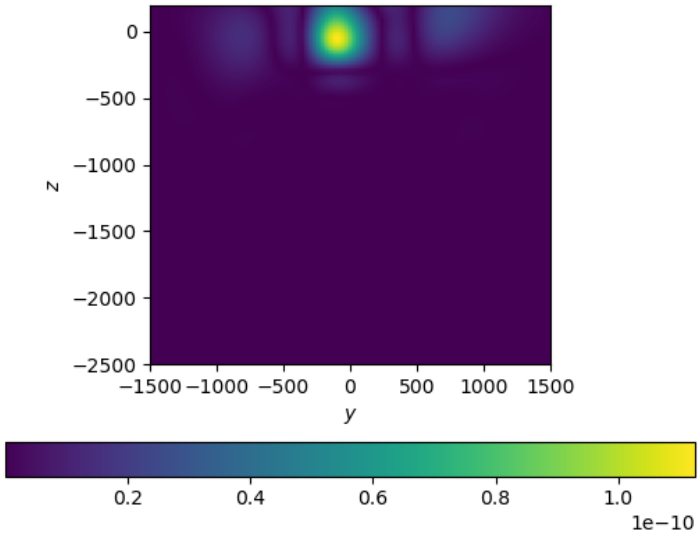


Figure 17: Norm of $\vec{E_x}$ (V/m) on slice $x = 400$.

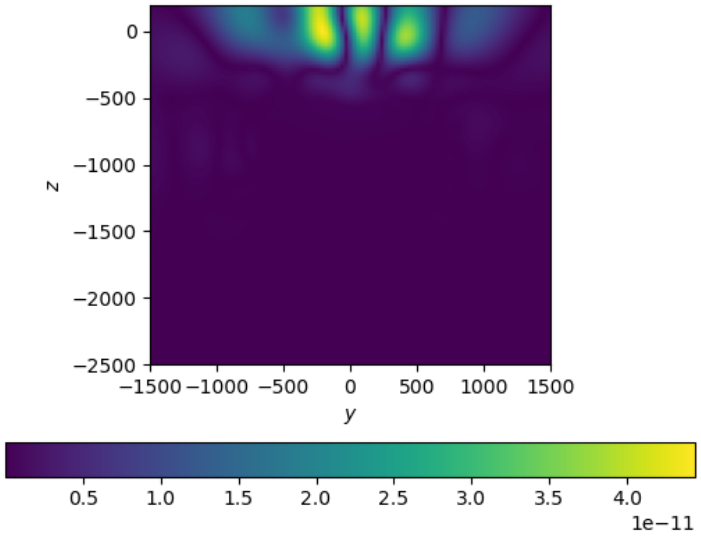


Figure 18: Norm of $\vec{E_y}$ (V/m) on slice $x = 400$.

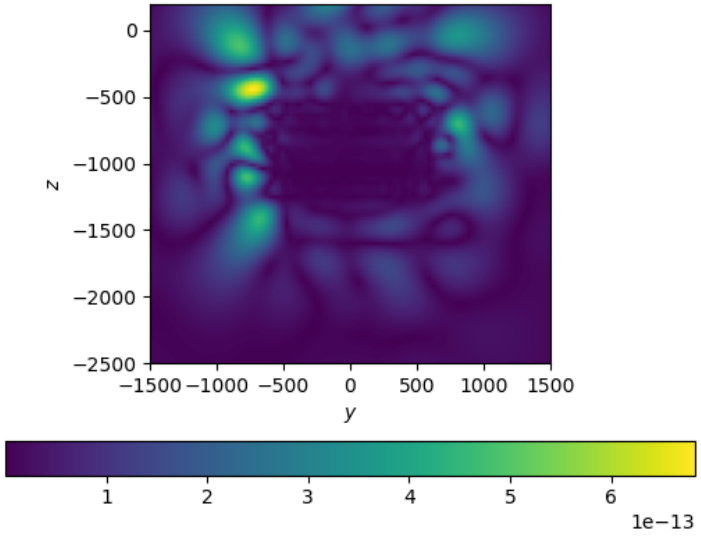


Figure 19: Norm of $\vec{E_z}$ (V/m) on slice $x = 400$.

We can see that not many refinements are made around the block. Most refinements are near the edges and the corners of the block, which can be seen in Fig. 14 and 15. Finally we note that most refinements have taken place around the transmitter and the receivers, which is visible in Fig. 16. With regards to the transmitter, this is because it is modeled as a point source rather than a volume, which causes the solution to fluctuate heavily around it. The receivers also suffer from being modelled as points, although the solution changes a lot less compared to the area

around the transmitter. The adaptive algorithm further refines the mesh around these points to more accurately model the changes in the electrical field.

In Fig. 17 up until 19 the electric field on a section of the domain is displayed. Fig. 17 and 18 clearly show that the electric field is largest in magnitude around the transmitter. Fig. 19 provides more interesting information. It can be seen that at the left and right edges ($y = -500$ and $y = 500$ respectively) of the block, the field has a steep gradient. It increases from a relatively low to a high value due to the increase in resistivity and then decreases again inside the block where the resistivity remains constant. The top- and bottom of the block are located at $z = -800$ and $z = -1000$ respectively. However, the figure suggests that the block has a greater height; the steep gradients of the field are located around $z = -500$ and $z = -1200$. This is most likely because of the interaction of fields between the transmitter as the main source and the scattering field that comes from block itself. This explains why the algorithm also refines above and below the block.

We now consider the result of the same block, but rotated 15 degrees around the z-axis. Again the adaptive algorithm was run on a basic octree mesh with minimum cell width of 50. Slices of the final converged mesh are shown in Fig. 20 and 21.

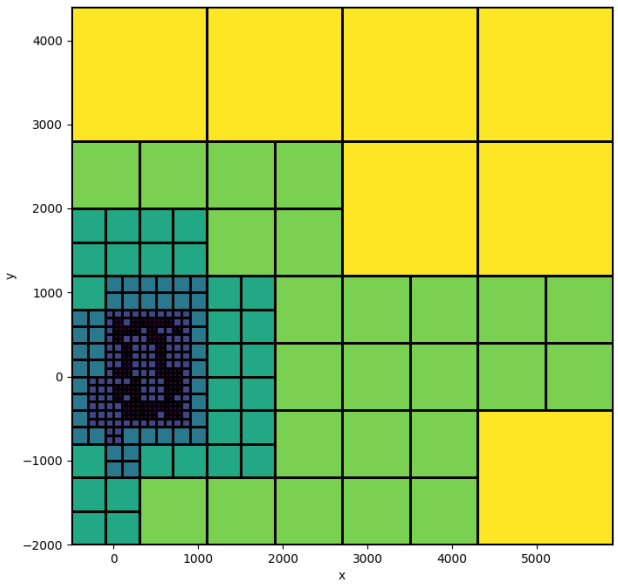


Figure 20: 2D slice at $z = -900$ of the converged octree mesh for the rotated block model.

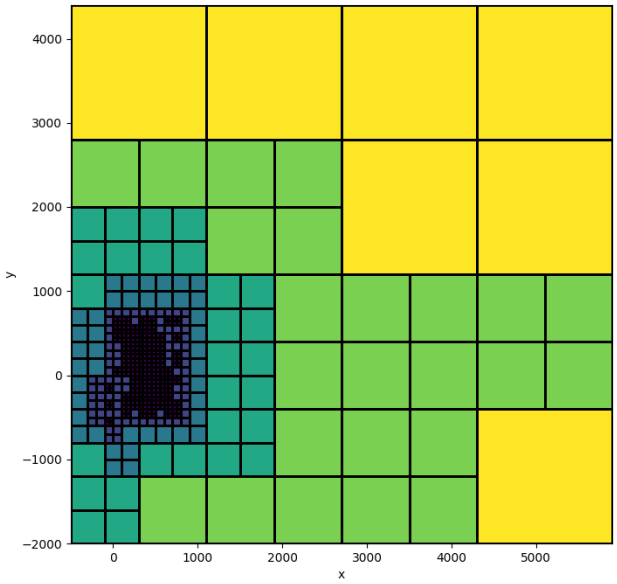


Figure 21: 2D slice at $z = -800$ of the converged octree mesh for the rotated block model. This slice shows the top face of the block.

We can see that the refinements placed by the algorithm are more random than with the regular block model. In Fig. 20 and 21 we see random spots around the block that have been refined. The places have probably been refined because the staircase effect has led to larger errors around those areas. Just as with the regular block model, the mesh has been refined around the transmitter and the receivers leading to a similar slice as Fig. 16.

The final model we used is a sphere model with an origin at $(1000, 300, -1200)$ and a radius of $r = 600$. We again ran the adaptive algorithm using a minimum cell width of 50. Slices of the final octree mesh that came out of the iterative scheme for the sphere are shown in Fig. 22 up until 24.

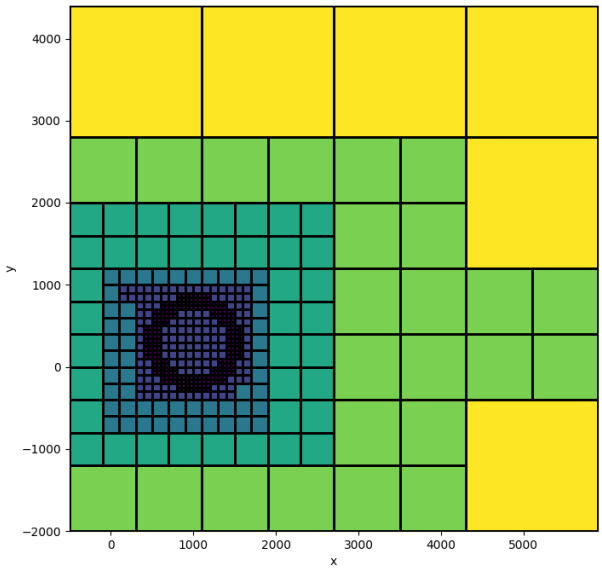


Figure 22: 2D slice at $z = -1525$ of the converged octree mesh for a perfect sphere with $r = 600$ and origin placed at $(1000, 300, -1200)$.

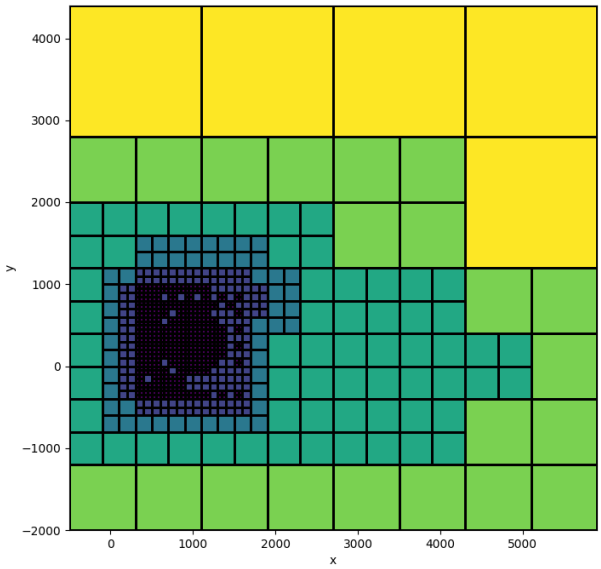


Figure 23: 2D slice at $= -625$ of the converged octree mesh for a perfect sphere with $r = 600$ and origin placed at $(1000, 300, -1200)$. This slice shows the top of the sphere.

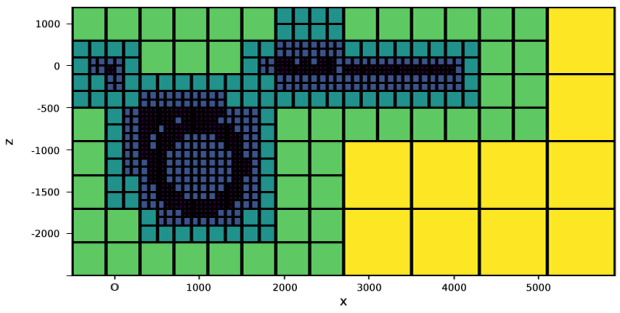


Figure 24: 2D slice at $y = -25$ of the converged octree mesh for a perfect sphere with $r = 600$ and origin placed at $(1000, 300, -1200)$.

As visible in the figures, the mesh is at its finest on the surface of the sphere, where the electric field rapidly changes value. Fine cells were required to accurately model this steep gradient. No additional refinements are placed inside the sphere and in the rest of the domain where it would be unexpected as the the electric field should remain almost constant there. However, two remarkable observations came out of this experiment that also appeared in the experiments for the other shapes.

First of all, the convergence for the sphere in Fig. 13 shows an almost root-like behaviour. The sequence for the convergence is as follows. In the first iterations, the refinements have a large effect on the solution of the electric field. It will refine the locations where refinements are most

needed, namely near the surface of the sphere. The solution of the next iteration will therefore be significantly different from the previous iteration. The refinements in the next iteration will still be near the surface of the sphere, but its effect on solution will reduce since the refinements that were most necessary are already added. The consequence is that the relative difference between the current and previous iteration will become much lower. However, no threshold was set in order to flag cells for refinement. If this were the case, the number of refinements in each iteration would decrease and we would expect the convergence plot to show linear behaviour. The error estimator flags 5% of the cells with the largest errors in each iteration. If these cells are at their maximum refinement level, they will not be refined further. Despite this, the number of actual added refinements each iteration decreases at a lower rate compared to the situation where a threshold would be defined. Because of this refinement percentage, the error estimator starts to flag cells that are not so much near surface of the sphere compared to previous iterations. These new refinements somewhere in the domain will cause a larger change of the solution than refinements near the surface of the sphere that are already accompanied by fine cells. That is why the relative difference in later iterations decreases at a much lower rate than in previous iterations and can even increase sometimes. This is shown in Fig. 12. The complete convergence behaviour is difficult to predict as Fig. 12 also shows. Near the end, the average relative difference suddenly drops fast.

The second remarkable observation is shown in Fig. 23 and 24: It can be seen that in the left half of the top of the sphere a lot of refinements are added. These refinements quite noticeably point in the direction of the transmitter. The reason for this is that the electric field starts at a relatively high value in the source, then quickly falls to a significantly lower value when moving away from the source. At the surface of the sphere, the electric field increases again in value because of the higher resistivity. However, this increase is larger at points of the surface that are closer to the transmitter in terms of Euclidean distance. The farther the field gets from the source, the weaker the response of the sphere in these regions. This is why there is a steeper gradient of the field between the surface and the transmitter than for example at the bottom of the sphere. Finer cells are required to properly model this steep gradient.

4.2 Convergence of the Adaptive Algorithm on a Realistic Model

In order to determine the use of our algorithm we tested the algorithm on a more realistic model that could be used in the field of geophysical sciences. The model presented in this report is an artificial model based on a sub-section of the Perth Basin located in the west of Australia [14]. We tested the convergence of the algorithm applied to this model. The domain included a large body of water starting at -500 m below sea-level. The seafloor starts at about -2200 m. The model extended the depth of the seafloor to -5500 m below sea-level. The domain is shown in Fig. 25 and 26. Fig. 26 also shows the locations of the transmitter and receivers. The transmitter had a frequency of 1 (Hz).

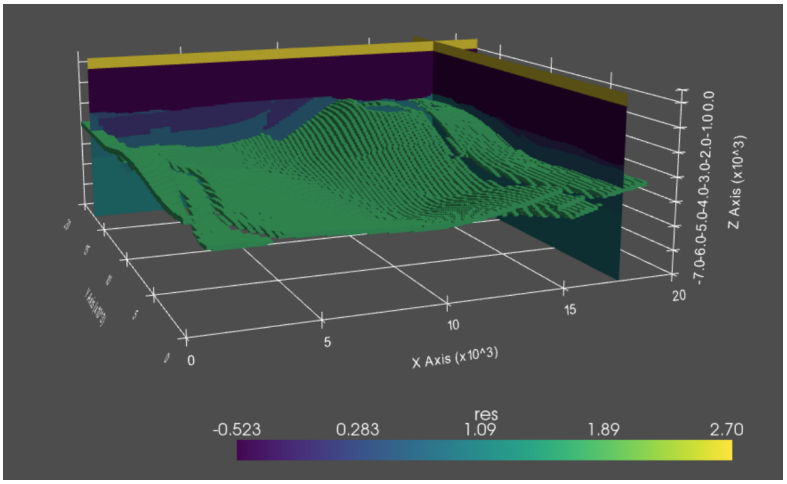


Figure 25: Three dimensional representation of the sea-floor of the large artificial model, showing the resistivity values [15].

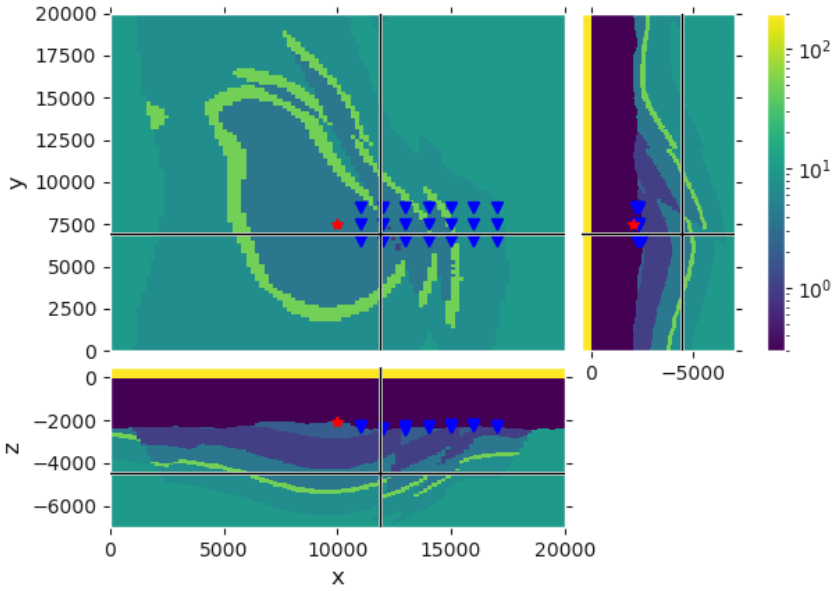


Figure 26: Two dimensional slices of the large artificial model, showing both the resistivity values and the locations of the transmitter, depicted by a star, and the receivers, depicted by triangles.

A minimum cell width of 250 was used to generate and iterate the octree mesh. In Fig. 27 the convergence of the mesh is shown. The model converged in 6 iterations and again shows an almost root-like behaviour. The explanation for this is provided in section 4.1.

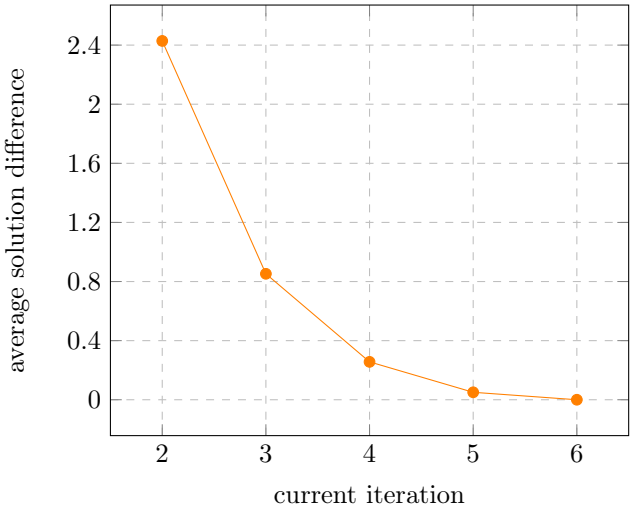


Figure 27: The convergence of the adaptive algorithm for the large artificial model. The average difference between the solutions of iteration i and $i-1$ is plotted against the iteration number i.

The final octree mesh can be observed in Fig. 28 up until 34. In Fig. 28 the seawater is shown, the mesh is allowed to be coarse here because the resistivity is constant. There are some refinements made along the direction of the transmitter, as the response of the electric field fluctuates more when it is closer to the transmitter. In Fig. 29 the seafloor is shown. The mesh is fine in this region as the resistivity suddenly changes from that of the seawater to the resistivity of the seafloor. This can be seen in Fig. 26. It causes a sudden change of the electric field, which requires fine cells to properly model. In Fig. 30 the bottom of the domain is shown. The mesh is allowed to be coarse here again because the field remains relatively constant.

From Fig. 31 up until 33 x-slices of the mesh are shown. As expected, most refinements are added near the transmitter located at $(10000, 7500, -2000)$. This is because the electric field starts at a high value and decreases rapidly when the field moves away from the transmitter. Fine cells are required to model this steep gradient. In Fig. 33 not as many fine cells are required as the domain is relatively far from the transmitter. Furthermore, it can be observed that more refinements are added below the seafloor than above the seafloor, stretching to -4000. We refer back to Fig. 26. It can be seen that this is a region where the resistivity varies quite a lot compared to the domain above -2000, causing the electric field to fluctuate heavily in these regions. Fig. 34 shows a y-slice of the domain. Again, refinements are shown in regions where it is expected. At $x = 18000$ some refinements in the vertical direction are added, which are located around the most far-right receiver. The adaptive algorithm provides us with an octree mesh that is in agreement with its hypothetical expectation.

The x-,y- and z-components of the electric field $\vec{E}$ are depicted in Fig. 35 up until 40. It is immediately clear that the field is largest in value around the transmitter. Because of this, the gradient of the field at the boundary from the seawater to the seafloor far away from the transmitter is flatter and therefore not as visible (though indeed present). In the figures, $\vec{E_y}$ and $\vec{E_z}$ show most visibly what has been described before. When the resistivity increases at the seafloor, suddenly the electric field increases as well. This response of the field is most dramatic near the transmitter. Further down into the ground the electric field increases and decreases as well due to the changing resistivity. This can be seen in Fig. 37 and 39, albeit not very clearly. Again, this is because the regions are relatively far from the transmitter, thus the gradients are

flatter compared to the fluctuations of the electric field near the transmitter.

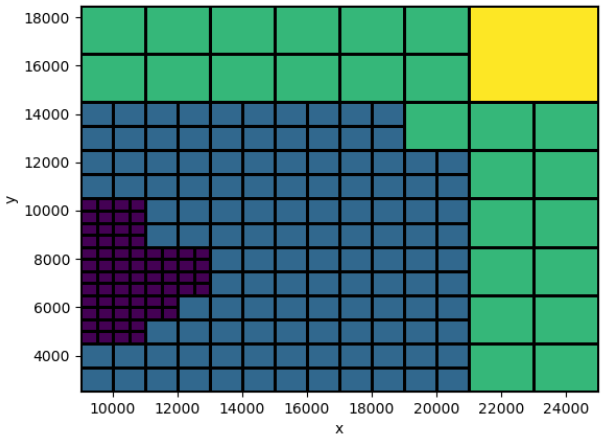


Figure 28: 2D slice of the octree mesh for the large artificial model. Slice is located in the seawater at $z = -875$.

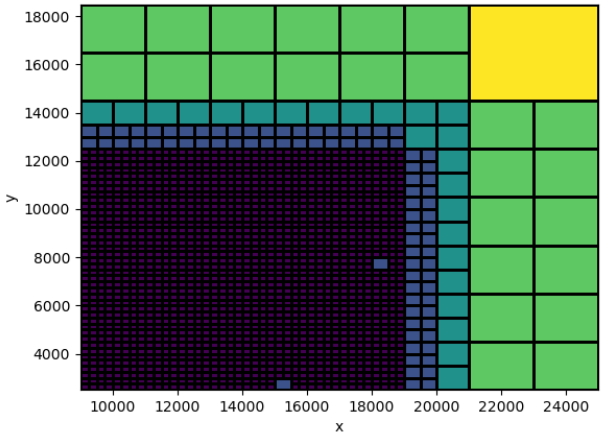


Figure 29: 2D slice of the octree mesh for the large artificial model. Slice is located on the seafloor at $z = -2125$.

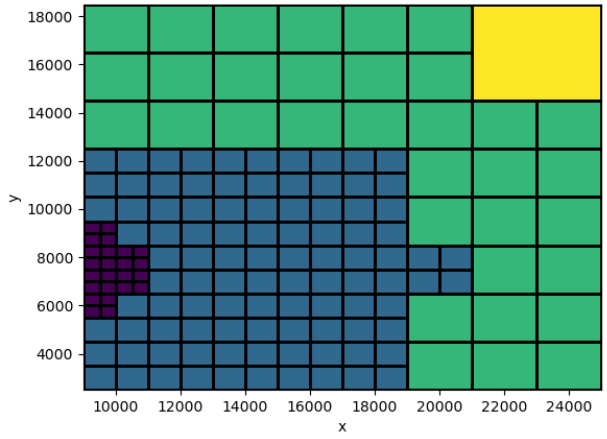


Figure 30: 2D slice of the octree mesh for the large artificial model. Slice is located in the ground at $z = -5375$.

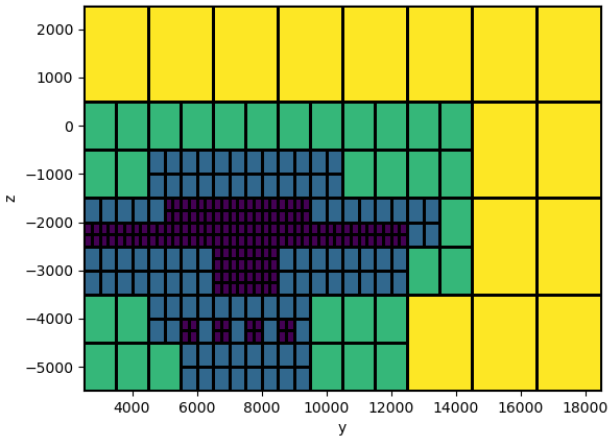


Figure 31: 2D slice of the octree mesh for the large artificial model. Slice is located at $x = 9125$.

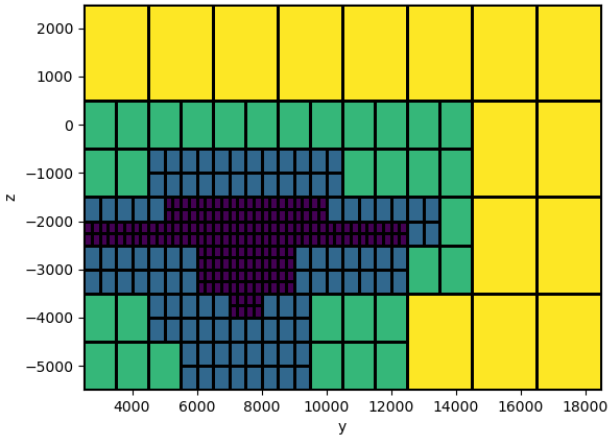


Figure 32: 2D slice of the octree mesh for the large artificial model. Slice is located at $x = 9875$ near the transmitter.

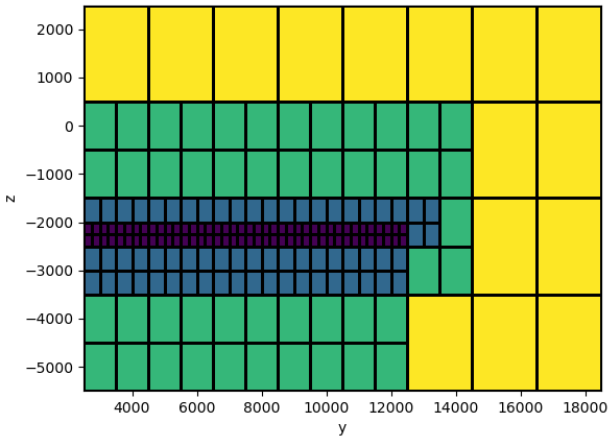


Figure 33: 2D slice of the octree mesh for the large artificial model. Slice is located at $x = 14125$.

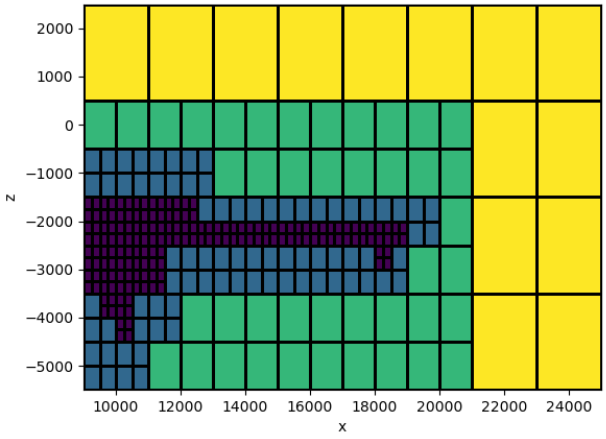


Figure 34: 2D slice of the octree mesh for the large artificial model. Slice is located at $y = 7375$ near the transmitter.

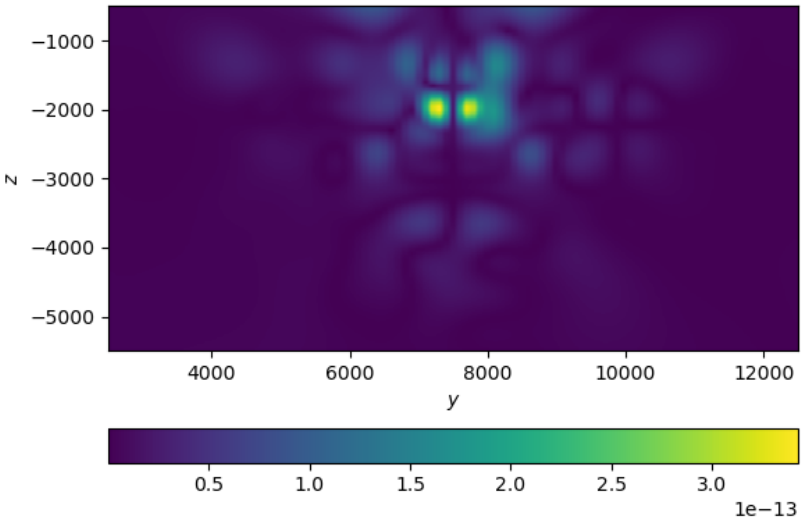


Figure 35: Norm of $\vec{E_x}$ (V/m) on slice $x = 9000$.

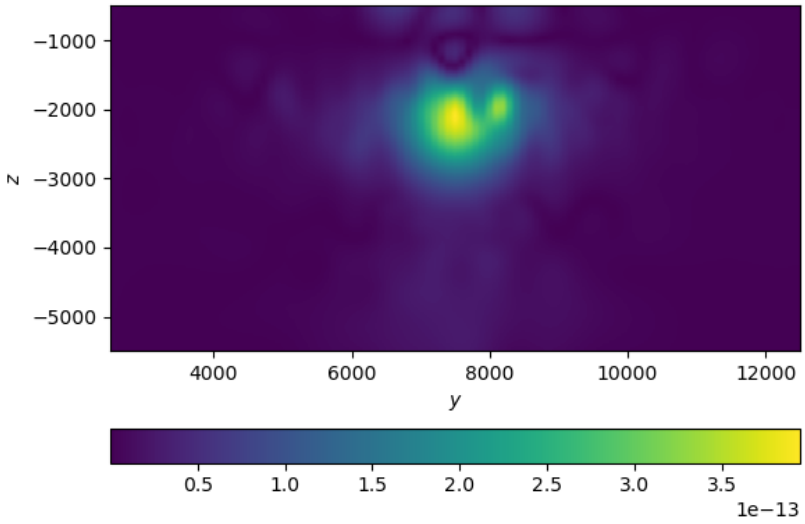


Figure 36: Norm of $\vec{E_y}$ (V/m) on slice $x = 9000$.

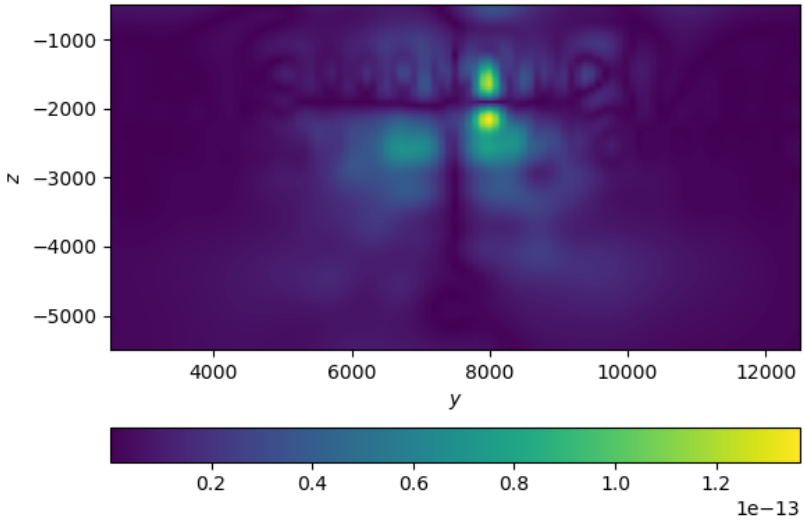


Figure 37: Norm of $\vec{E_z}$ (V/m) on slice $x = 9000$.

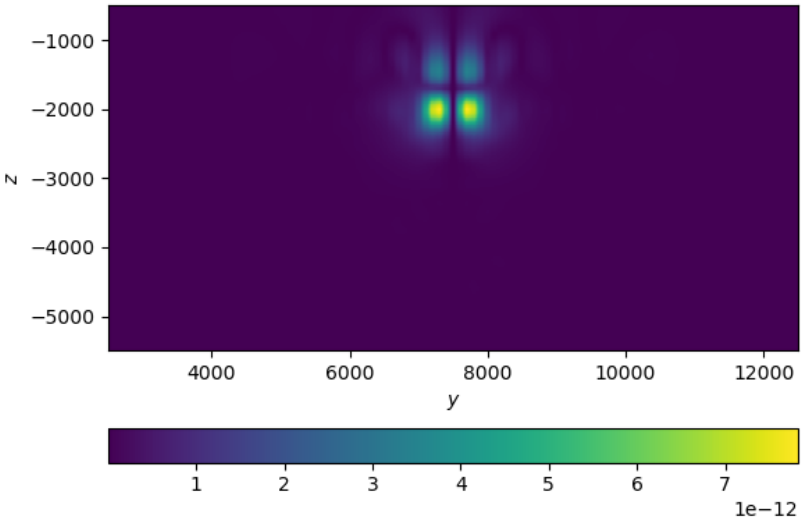


Figure 38: Norm of $\vec{E_x}$ (V/m) on slice $x = 10000$.

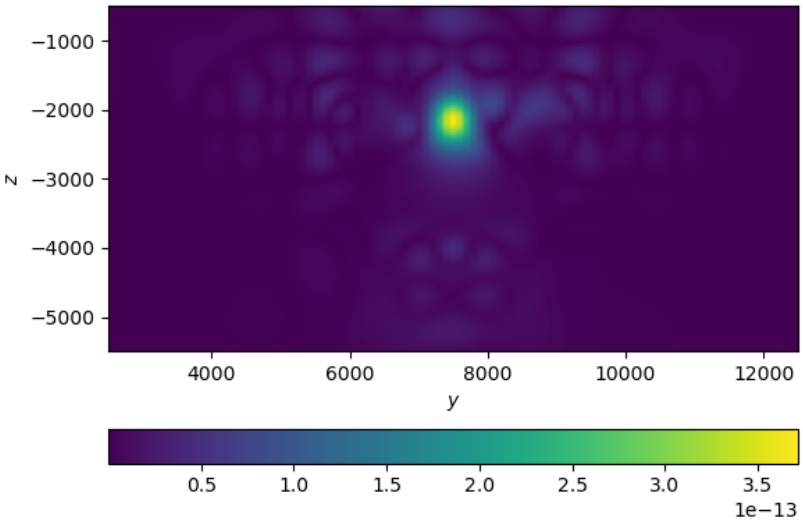


Figure 39: Norm of $\vec{E_y}$ (V/m) on slice $x = 10000$.

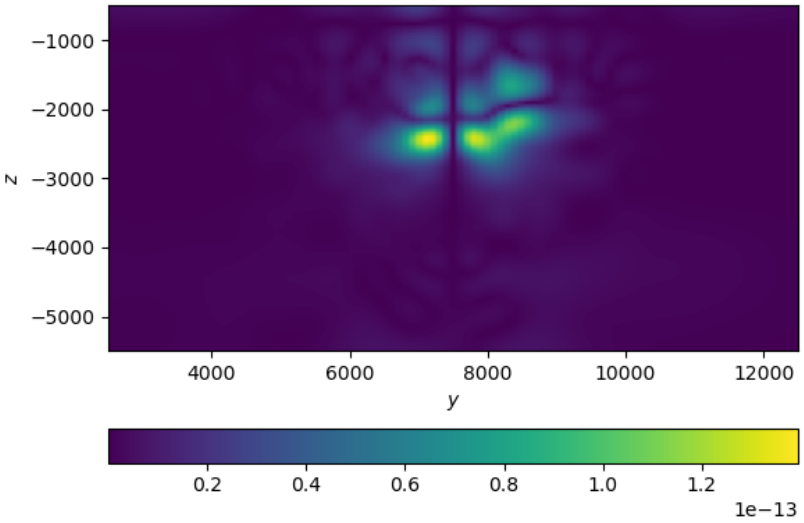


Figure 40: Norm of $\vec{E_z}$ (V/m) on slice $x = 10000$.

4.3 Comparing the Solution Accuracy of an Adapted Mesh With a Benchmark

The first two experiments served to analyze the convergence of the adaptive algorithm, but presence of convergence does not imply that the actual numerical solutions are accurate. As it should be possible to run the adaptive algorithm on realistic problems, the solver should deliver accurate numerical solutions. Therefore, in this experiment we investigate whether the adaptive algorithm can deliver a good numerical approximation. To investigate the accuracy of the solution, we compare the adaptive algorithm with a triply uniform tensor mesh.

The set up of the experiment is as follows: The earlier mentioned basic block model is solved with the conventional solver using a uniform tensor mesh as well as with the adaptive algorithm using an octree mesh for a given minimum cell width. The tensor mesh had a cell width of 50, which was also the minimum cell width for the octree mesh. Both solutions were interpolated and then compared to each other in locations of interest. This comparison assumes that the tensor mesh is the better approximation of the exact solution. The tensor mesh solution is thus taken as a benchmark solution. The reason for this is that the tensor mesh is either just as fine as the octree mesh or finer everywhere in the domain. Furthermore, the tensor mesh does not suffer from interpolation errors that the octree mesh has to make in order to approximate a solution at a point that does not exist in the grid. This situation occurs on a coarse-fine cell interface (the transition from a coarser cell to four finer cells). Comparing the relative error between the adaptive algorithm and the conventional solver resulted in the following plots.

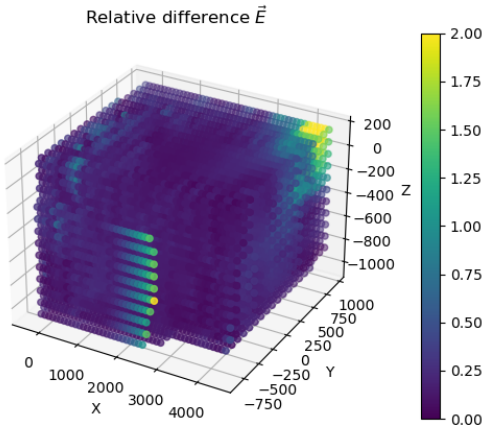


Figure 41: Relative difference in the electric field between the octree- and tensor mesh.

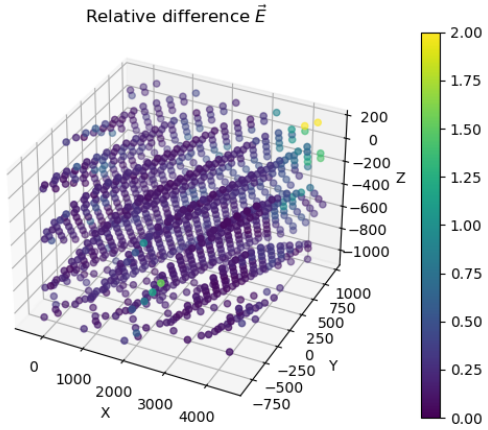


Figure 42: Relative difference in the electric field between the octree- and tensor mesh.

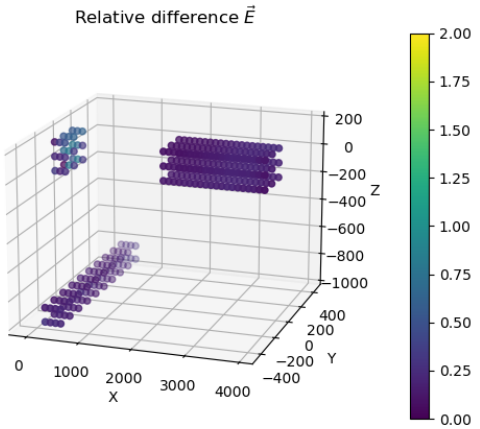


Figure 43: Relative difference in the electric field between the octree- and tensor mesh. The relative difference is limited to the locations of the transmitter, receivers and block.

The formula used to compute the relative error is as follows:

$$\epsilon = \frac{||\vec{E}_{\text{oct}} - \vec{E}_{\text{ten}}||_2}{||\vec{E}_{\text{ten}}||_2} \tag{12}$$

Where ϵ is the relative error, $||\vec{x}||_2$ is the L_2-norm of the vector $\vec{x}$, $\vec{E}_{\text{oct}}$ is the numerical solution for the electric field using the octree mesh and $\vec{E}_{\text{ten}}$ is the solution for the electric field using the tensor mesh.

Fig. 41 and 42 show that the relative difference is below 20% almost everywhere in the domain, except near the boundaries and in the transmitter itself. The large difference in these regions is because of discretization and interpolation errors. The octree mesh has coarse cells near the boundaries of the domain, thus the solution will not be as accurate here compared to the tensor mesh. This is also the case for the transmitter, where the electric field has a steep gradient. A lot of fine cells are required to accurately reflect this gradient. That is why the difference between the octree mesh and tensor mesh is more than 50% here. The average relative difference was 19.2% in the whole domain and 15.8% around the receivers. It can be concluded that the tensor mesh is much more accurate. However, the price to pay is computation time. The octree mesh had 2,808 cells while the tensor mesh had 262,144 cells. Computation time scales exponentially with the number of cells for the direct solver of `SimPEG` as will be shown in section 4.4. From this perspective, it can be concluded that the octree mesh is much more efficient than the tensor mesh.

4.4 Scalability of Solvers With Increasing Number of Mesh Cells

In this experiment we investigated how the computation time of solving the electric field on different meshes scales with the number of cells in the mesh. By using various minimum cell sizes, the number of cells in the mesh can be controlled. We ran the adaptive algorithm on the basic block model with various minimum cell sizes and noted the number of cells in the final mesh. Further, we noted the duration of a single iteration in the adaptive algorithm, which contains the duration of computing the electric field. We also did the same using `em3d` using a tensor mesh so that we can compare the time complexity of both algorithms. The results of these experiments are shown in Fig. 44 and 45.

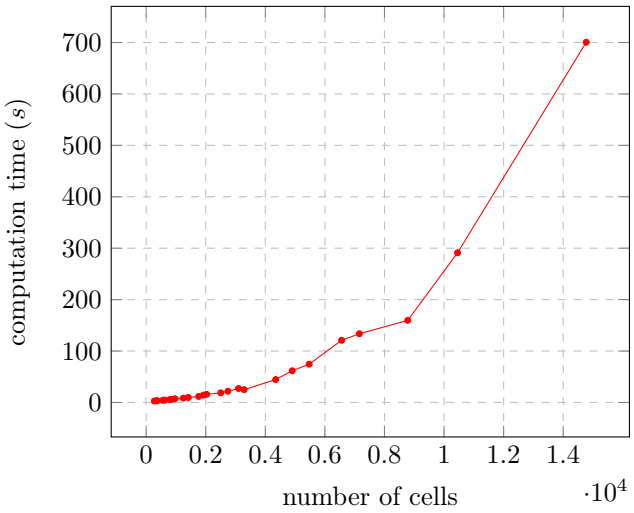


Figure 44: The computation time of the octree solver against the number of cells that are used in the octree mesh. The times are noted for a single iteration.

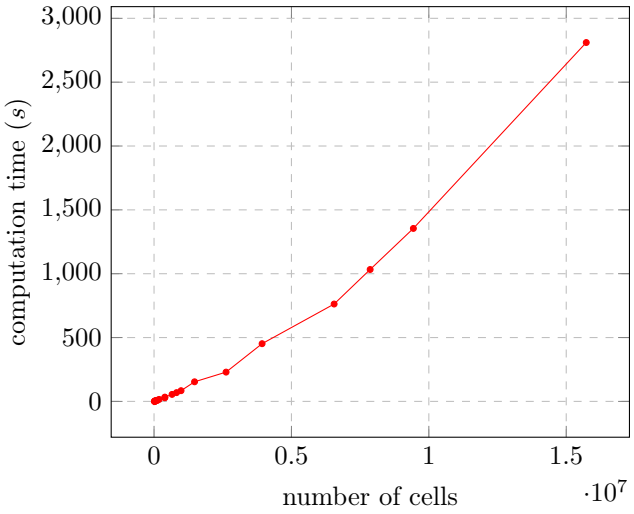


Figure 45: The computation time of the conventional solver `emg3d` against the number of cells that are used in the tensor mesh.

When we look at Fig. 44 we see that there is an almost exponential increase in the computation time given the number of mesh cells. On the other hand `emg3d` shows an almost linear dependence between the computation time and the number of cells in Fig. 45. The question remains whether the adaptive algorithm has exponential complexity due to the `SimPEG` solver or due to the error estimation and interpolation. To investigate this we repeated the experiment of `emg3d` with `SimPEG` noting only the duration of computing the electric field. Fig. 46 shows the computation times of `SimPEG` against the number of cells in the mesh.

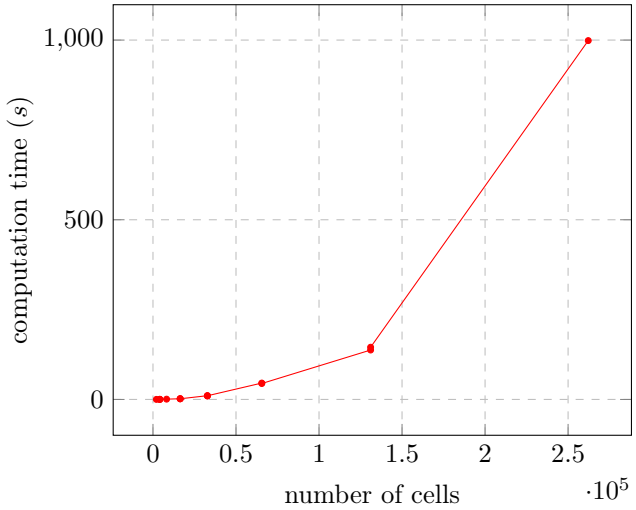


Figure 46: The computation time of the `SimPEG` solver against the number of cells that are used in the tensor mesh.

From Fig. 46 we see an exponential dependence between the number of cells and the computation time. Thus we could argue that the exponential growth in computation time of our adaptive algorithm we see in Fig. 44 is explained by the time complexity of the method we use to solve the numerical problem.

Further, Fig. 44 and Fig. 46 only show the computation times of a single iteration, so the total mesh refinement time is at least 6 times larger due to the 6 necessary iterations to converge. We could thus conclude that the adaptive algorithm is rather inefficient in generating a mesh.

However, it is important to note that the octree mesh naturally has much less cells than the tensor mesh of `emg3d`. Because of this, the exponential dependence between the computation time and the number of cells is not quite as problematic. Keeping this in mind, `emg3d` performs faster than the adaptive algorithm as a whole, but generating a solution on the refined octree mesh could be faster than `emg3d`.

In an ideal case `emg3d` could be adapted such that it could run on octree meshes. In that case, the adaptive algorithm could use `emg3d` to compute the electric field in each iteration. Possibly this could lead to a linear complexity of the complete adaptive algorithm depending on the number of cells in the meshes.

5 Discussion and Recommendations

In the results of the experiments we saw multiple things. First, we saw that for several test cases, both including basic shapes and a larger model, the algorithm does converge to a final octree mesh. These meshes were in agreement with the expected outcomes based on theory into the behaviour of the electric field under influence of transmitters, receivers and varying resistivity in the domain. The convergence of the algorithm often displays a root-like behaviour. It was explained that this was due to the constant refinement percentage, causing to refine too few cells in the first iterations and more cells than necessary in the last iterations. At locations where these refinements are superfluous, often in coarse regions, it can influence the field solution more than in regions that are accompanied by fine cells. As a consequence, the convergence rate slows down after a certain number of iterations and the relative difference can sometimes even increase before it starts to decrease again.

The algorithm could be optimized by analyzing and finding an error threshold that could replace the refinement percentage. If the error estimate exceeds this threshold, the region should be refined. It is expected that this will improve the convergence rate, since only regions are refined that actually need this refinement in each iteration. The algorithm would in this scenario most likely show linear behaviour.

Secondly, from the results it can be concluded that the converged octree mesh is efficient regarding the solving time of only the final mesh, as the octree mesh contains approximately 93 times less cells than the tensor mesh. The trade-off is however that the solution accuracy is not as high as with the tensor mesh, as the solution we saw differed on average 15.8% from the benchmark solution around the receivers.

Furthermore, we saw that the computation times grow exponentially with the number of cells in the mesh for the adaptive algorithm. This is also the case for the tensor mesh using the `SimPEG` solver. `emg3d` scales linearly with increasing problem size, because it uses a multi-grid iterative solver rather than a direct solver. Therefore, if one were to generate synthetic data on a model only once, `emg3d` should be used. This is less expensive than our algorithm that has to iteratively adapt the mesh in often more than six iterations. However, once an optimized octree mesh has been obtained from the algorithm, further numerical problems can be solved more efficiently with this mesh, given that the boundary conditions and initial conditions of the domain remain unchanged. By reusing the octree mesh for multiple experiments, the average computation time for each experiment would be lower than running the complete refined grid of `emg3d` for each experiment. In that case, it would be worth using more computation time to generate an efficient octree mesh once with our algorithm.

One aspect that can be explored is the maximum allowed size for a cell. In the current algorithm a cell can be as large as possible, while this may result in large interpolation errors. The algorithm could be adjusted to refine the mesh to at least a certain level, such that the aspect ratio between the largest cells and the smallest cells is not too great. This might increase the number of cells in the mesh, but could lead to a higher accuracy in the solution.

A final possibility is to employ parallel programming to improve the performance of the adaptive algorithm. The error estimator is currently applied to each cell center in a linear loop, but this loop could be parallelized as each loop iteration is dependent of other iterations. Further, also the interpolation at different coordinates could be parallelized. While parallelizing may not remove the exponential time complexity of the algorithm, the adaptive algorithm could become faster for reasonable minimum cell widths.

6 Conclusion

The questions we asked in this study were how we could develop an adaptive meshing algorithm and how such an adaptively refined mesh compares to uniform meshes used by for example `emg3d`. For this comparison we were interested in the difference between computation time and the solution accuracy on the different mesh types.

We developed an algorithm that makes use of `SimPEG` and the ORESD method applied to the curl operator of the electric field. The adaptive mesh starts off relatively coarse with some initial refinements. During the iterations of the algorithm, the mesh adapts itself by adding refinements at locations the error estimator points out until the mesh is at its most efficient in terms of ratio between cells and solution accuracy.

The algorithm delivers an octree mesh in six to fifteen iterations depending on the cell width, domain size and conductivity/resistivity mapping. This final mesh is in accordance with what is expected based on the theory of electromagnetic methods. The solution computed on the octree mesh is not as accurate as the solution computed on a tensor mesh, but the octree mesh does contain less cells which leads to more efficient solving times.

However, iteratively refining the mesh does take more time and the complexity of the algorithm scales exponentially with the number of mesh cells. `emg3d` for example scales linearly with the number of cells, which leads to shorter computation times if we consider the complete running time of our algorithm.

Therefore, if forward modelling is performed only once in a domain, it would be preferred to use a conventional tensor mesh, which makes use of iterative solvers. On the other hand, if forward modelling has to be performed multiple times in the same domain, it would save a lot of computation time to let the adaptive algorithm generate an efficient mesh once and then use this mesh for further modelling.

The largest issue coming from the developed algorithm is the computation time required to generate an efficient octree mesh. Therefore, several optimizations are required in order to let it compete with conventional mesh generators. One of the major factors that leads to an improved algorithm is to analyze and find a threshold for the ORESD error estimator. This will increase the convergence rate of the iterative scheme as all regions that require refinement will be refined in one iteration and superfluous refinements are avoided. Another issue is the interpolation error coming from large cells near the edges of the domain. Setting a limit to this cell size trades a larger computational cost for higher accuracy. Lastly, since the algorithm showed great potential for parallelism, implementing parallelism can possibly decrease computation time as well.

References

[1] K. Vozoff. "Electromagnetic Methods in Applied Geophysics". In: *Geophysical surveys* 4.1 (Sept. 1980), pp. 9–29. ISSN: 1573-0956. DOI: 10.1007/BF01452955.

[2] João Pedro A. Bastos and Nelson Sadowski. *Electromagnetic Modeling by Finite Element Methods*. CRC Press, Apr. 2003. ISBN: 978-0-203-91117-4.

[3] Randall L. Mackie, J. Torquil Smith, and Theodore R. Madden. "Three-Dimensional Electromagnetic Modeling Using Finite Difference Equations: The Magnetotelluric Example". In: *Radio Science* 29.4 (1994), pp. 923–935. ISSN: 1944-799X. DOI: 10.1029/94RS00326.

[4] S. H. Ward and G. W. Hohmann. "Electromagnetic Theory for Geophysical Applications, in Electromagnetic Methods in Applied Geophysics—Theory". In: *Society of Exploration Geophysicists* 1 (1988), pp. 131–311.

[5] T. Weiland. "Eine Methode zur Lösung der Maxwellschen Gleichungen für sechskomponentige Felder auf diskreter Basis: Archiv für Elektronik und Übertragungstechnik". In: *leibniz* 31 (1977), pp. 116–120. URL: leibniz-publik.de/de/fs1/object/display/bsb00064886_00001.html.

[6] K. Ho-Le. "Finite Element Mesh Generation Methods: A Review and Classification". In: *Computer-Aided Design* 20.1 (Jan. 1988), pp. 27–38. ISSN: 00104485. DOI: 10.1016/0010-4485(88)90138-8.

[7] Pouya Sadeghi Farshbaf, Mohammad Mahdi Khatib, and Hamid Nazari. "Solid Meshing of 3D Geological Model in Finite Element Analysis: A Case Study of East Azerbaijan, NW Iran". In: *Modeling Earth Systems and Environment* 2.1 (Dec. 2015), p. 12. ISSN: 2363-6211. DOI: 10.1007/s40808-015-0066-6.

[8] G. Legrain, R. Allais, and P. Cartraud. "On the Use of the Extended Finite Element Method with Quadtree/Octree Meshes". In: *International Journal for Numerical Methods in Engineering* 86.6 (2011), pp. 717–743. ISSN: 1097-0207. DOI: 10.1002/nme.3070.

[9] Dieter Werthmüller, Wim A. Mulder, and Evert C. Slob. "Emg3d: A Multigrid Solver for 3D Electromagnetic Diffusion". In: *Journal of Open Source Software* 4.39 (July 2019), p. 1463. ISSN: 2475-9066. DOI: 10.21105/joss.01463.

[10] Rowan Cockett et al. "SimPEG: An open source framework for simulation and gradient based parameter estimation in geophysical applications". In: *Computers Geosciences* 85 (2015), pp. 142–154. ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2015.09.015. URL: http://www.sciencedirect.com/science/article/pii/S009830041530056X.

[11] Mark A. Yerry and Mark S. Shephard. "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique". In: *International Journal for Numerical Methods in Engineering* 20.11 (1984), pp. 1965–1990. ISSN: 1097-0207. DOI: 10.1002/nme.1620201103.

[12] Giovanni Lapenta. "A recipe to detect the error in discretization schemes". In: *International Journal for Numerical Methods in Engineering* 59.15 (2004), pp. 2065–2087. DOI: 10.1002/nme.969. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.969. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.969.

[13] Randolph E. Bank and Jinchao Xu. "Asymptotically Exact A Posteriori Error Estimators, Part II: General Unstructured Grids". In: *SIAM Journal on Numerical Analysis* 41.6 (2003), pp. 2313–2332. DOI: 10.1137/S0036142901398751. eprint: https://doi.org/10.1137/S0036142901398751. URL: https://doi.org/10.1137/S0036142901398751.

[14] emg3d Gallery. *Simulation Example*. URL: https://emsig.github.io/emg3d-gallery/gallery/tutorials/simulation.html.

[15] Dieter Werthmüller and Evert Slob. *2. GemPy-II: Perth Basin*. URL: https://emsig.github.io/emg3d-gallery/gallery/interactions/GemPy-II.html.