AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FIELD OF SCIENCE:
Technical Sciences
SCIENTIFIC DISCIPLINE:
Information and Communication Technology

DOCTORAL THESIS

Efficient algorithms for three-dimensional computational mesh generations and air pollution simulations based on hypergraph grammars

Author: Krzysztof Podsiadło

Supervisor: prof. dr hab. Maciej Paszyński

Completed in:
AGH University of Science and Technology in Kraków
Faculty of Computer Science, Engineering and Communication
Institute of Computer Science

Krakow, 2022

This work has been supported by National Science Centre grant HARMONIA no. 2017/26/M/ST1/ 00281. I would also like to thank Prof. Keshav Pingali from The University of Texas at Austin for hosting my visit at Oden Institute, Prof. Rafael Montenegro-Armas for hosting my visit at The University of Las Palmas de Gran Canaria, and Dr Albert Oliver-Serra for fruitful scientific collaboration.

I wish to dedidacate this work to all who have helped me to create it:
To the good God whose mighty blessing still surprises me.
To my soon-to-be wife Gosia who shows me what is truly important.
To prof. Maciej Paszyński, for his tremendous help and patience.
To my parents, family and friends for their gratuitous support.

ABSTRACT

This thesis aims to design, implement, and test a new class of graph-grammar-based algorithms for the generation of computational meshes and air pollution simulations. Our algorithms will employ the finite element method solver based on the advection-reaction-diffusion formulation. The algorithms will use the graph representation of the computational mesh adaptively built with the triangular finite elements, constructed based on the terrain topographic data. In particular, we express by the graph-grammar productions the process of generation of the computational mesh approximating the terrain structure, the generation of three-dimensional tetrahedral meshes filling the atmosphere on top of the terrain mesh, and the graph-grammar productions expressing the iterative solver algorithm. The expression of the mesh generation problem by graph-grammar production allows for parallel processing of the computational grid.

Computer simulations solving non-stationary problems such as the propagation of air pollutants using the finite element method are currently considered a hot scientific topic. This is because there is a need to reduce the computational cost of classical algorithms and the need to control the correctness of the computational process automatically. Expressing the computational process through the production of hypergraph grammar will allow solving both the problem of automatic control of the correctness of the generated computational meshes (the mesh generation process is described by means of several repetitive grammar productions), it will also reduce the computational cost (through the better location of computational tasks that can be processed in parallel). This work is a remedy for the well-known problem of the high computational cost of solvers for non-stationary simulations and the problems related to the generation of unstructured computational meshes.

This dissertation also proposes an algorithm for modifying the alternating-direction solver in order to take into account the irregular topography of the terrain. The proposed method opens new horizons for simulating atmospheric phenomena with the use of alternating-direction solvers.

STRESZCZENIE

Celem niniejszej rozprawy jest opracowanie i przetestowanie nowych algorytmów generacji siatek obliczeniowych i solwerów dla metody elementów skończonych w celu efektywnego przeprowadzania symulacji problemów propagacji zanieczyszczeń opisanych równaniami adwekcji-dyfuzji-reakcji. W szczególności wyrażenie procesu obliczeniowego, w tym generacji trójkątnych siatek obliczeniowych w oparciu o dane topograficzne terenu, generacja trójwymiarowych siatek czworościennych wyrażone zostanie poprzez produkcje gramatyki hipergrafowej. Umożliwi to uzyskanie większej dokładności i możliwości zrównoleglenia obliczeń. Ponadto planowane jest przeprowadzenie szeregu testów numerycznych na różnych architekturach maszyn równoległych.

Symulacje komputerowe do rozwiązywania problemów niestacjonarnych takich jak propagacja zanieczyszczeń powietrza za pomocą metody elementów skończonych są aktualnie uważane za gorący temat naukowy, ze względu na konieczność redukcji kosztu obliczeniowego algorytmów klasycznych oraz konieczność automatycznej kontroli poprawności procesu obliczeniowego.

Wyrażenie procesu obliczeniowego poprzez produkcję gramatyki hipergrafowej pozwolą rozwiązać zarówno problem automatycznej kontroli poprawności generowanych siatek obliczeniowych (proces generacji siatki opisany jest za pomocą kilku powtarzalnych produkcji gramatyki), pozwoli on również na redukcję kosztu obliczeniowego (poprzez lepszą lokalizacji tasków obliczeniowych które mogą być przetwarzane równolegle). Praca ta stanowi remedium na powszechnie znany problem dużego kosztu obliczeniowego solwerów do zadań niestacjonarnych, oraz problemów związanych z generacją niestrukturalnych siatek obliczeniowych.

W pracy zaproponowano również algorytm modyfikacji solwera zmienno-kierunkowego w celu umożliwienia uwzględnienia nieregularnej topografii terenu. Zaproponowana metoda stwarza nowe możliwości symulacji zjawisk atmosferycznych za pomocą solwerów zmienno-kierunkowych.

PUBLICATIONS

My scientific results are summarized in the following papers:

[1] Krzysztof Podsiadło, Albert Oliver Serra, Anna Paszyńska, Rafael Montenegro, Ian Henriksen, Maciej Paszyński, Keshav Pingali *Parallel graph-grammar-based algorithm for the longest-edge refinement of triangular meshes and the pollution simulations in Lesser Poland area*, **Engineering with Computers**, 37(4) (2021) 3857–3880. [1 kwartyl, IF: 7.963, 70 points]

[2] Maciej Paszyński, Leszek Siwik, Krzysztof Podsiadło, Peter Minev, *A massively parallel algorithm for the three-dimensional Navier-Stokes-Boussinesq simulations of the atmospheric phenomena*, **Lecture Notes in Computer Science** 12137 (2020) 102–117. [CORE A conference, 140 points]

[3] Barbara Barabasz, Stephen Barrett, Leszek Siwik, Marcin Łoś, Krzysztof Podsiadło, Maciej Woźniak, *Speeding up multi-objective optimization of liquid fossil fuel reserve exploitation with parallel hybrid memory integration*, **Journal of Computational Science** 31 (2019) 126–136. [1 kwartyl, IF: 4.97, 100 points]

[4] Krzysztof Podsiadło, Maciej Paszyński, Albert Oliver Serra, *Framework for topographic mesh generation and its application to the pollution simulations in Kraków area*, **KomPlasTech 2019 : XXVI International Conference on Computer Methods in Materials Technology** January 13-16, 2019, Zakopane

[5] Krzysztof Podsiadło, Albert Oliver Serra, Maciej Paszyński, *Framework for topographic mesh generation and its application to the pollution simulations in Kraków area*, **Computer Methods in Materials Science**, 19(1) (2019) 21–28. [40 points]

[6] Krzysztof Podsiadło, Marcin Łoś, Leszek Siwik, Maciej Woźniak, Maciej Paszyński, *An algorithm for tensor product approximation of three-dimensional material data for implicit dynamics simulations*, **IGA 2018 Integrating Design and Analysis** October 10–12, 2018, Austin

[7] Marcin Łoś, Judit Munoz-Matute, Krzysztof Podsiadło, Maciej Paszyński, Keshav Pingali, *Parallel shared-memory isogeometric residual minimization (iGRM) for three-dimensional advection-diffusion problems*, **Lecture Notes in Computer Science** 12143 (2020) 133–148. [CORE A conference, 140 points]

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

INTRODUCTION | STATE OF THE ART | MOTIVATION | THESIS

1 INTRODUCTION

Air pollution is of great interest these days. This is especially evident in the area of Kraków in Poland, as it is one of the most polluted cities in Europe [2]. People living there are becoming more and more aware of the problem, resulting in the creation of various Non-Government Organizations that are trying to improve air quality. Air pollution is increasing due to many factors, including traffic, climate, winter heating, city architecture, etc. An important research task is the ability to model atmospheric phenomena such as the propagation of pollutants over complex terrain.

1.1 MOTIVATION

The main thesis of this dissertation can be summarized as follows:

"It is possible to develop new, efficient algorithms for the generation of three-dimensional computational meshes and air pollution simulation solvers using the hypergraph grammar model. The developed new graph-grammar based mesh generation algorithms will be characterized by an automatic guarantee of the correctness of the computational mesh. They will allow for concurrent processing, including the mesh generation, mesh adaptation, as well as parallel execution of the advection-diffusion-reaction solver, using the developed model of concurrency based on hypergraph grammar productions. The developed new pollution simulation algorithms will be able to efficiently process a parallel computational process based on partition of the computational process into basic undividable tasks expressed by hypergraph grammar productions."

In other words, the aim of this dissertation is to design, implement and test new algorithms for the generation, and adaptation of computational meshes, as well as expressing the finite element method solvers in order to effectively simulate the process of propagation of pollutants described by the advection-diffusion-reaction equations. In particular, the expression of the computational process, including the generation of triangular

computational meshes based on topographic data of the terrain, and the generation of three-dimensional tetrahedral meshes, will be expressed through the production of hypergraph grammar. This will allow for greater accuracy and the possibility of parallelizing the calculations. Also, the global matrix is generated and processed element-wise, including vector-matrix multiplications of the iterative solver, without formulating the global matrix. In addition, it is planned to conduct a series of numerical tests on parallel shared-memory Linux cluster nodes.

1.2 STATE OF THE ART

1.2.1 *Adaptive finite element method computations over non-regular terrain*

Most computer-aided simulations begin by generating a domain mesh with a finite set of elements. For irregular geometries, triangular elements in two dimensions or tetrahedral elements in three dimensions are probably the most commonly used finite elements in engineering computations [7]. The construction of a computational mesh usually starts with an initial mesh. Then, it iteratively improves the mesh to obtain a final mesh in which we can solve our engineering problem with the required accuracy.

The process of refinement can generate so-called hanging nodes [12, 13]. In two dimensions, they represent an edge with one triangular element divided and a large continuous element on the other side. These nodes are difficult to handle because we have shape functions distributed over finite elements. For example, in the case of a hanging node, we have to deal with matching the approximation of a "small" shape function distributed over two damaged elements with the approximation of a "large" shape function distributed over a large uninterrupted element. In three dimensions, the situation becomes even more complicated because we can have an edge adjacent to two "small" corrupted tetrahedrons and to many (even hundreds) of continuous "large" elements.

The hanging node of a triangular element can be eliminated by splitting the element and connecting the hanging node to the opposite node. We need to perform this algorithm in a clever way because we do not want to end up with elongated elements where the Jacobians tend to zero. An interesting example of such an algorithm, which is considered a reference for

two-dimensional meshes, is Rivara's longest edge refinement algorithm [70, 72, 73].

In this paper, we express Rivara's two-dimensional algorithm using graph-grammar productions [75], and extend it to a model for generating three-dimensional quadrilateral meshes. We also propose graph-grammar productions expressing the stabilized finite element method for nonstationary simulations of advection-diffusion reactions. We incorporate the Crank-Nicolson time integration scheme and interface with the GMRES solver. Finally, we use this parallel graph-grammar system to simulate pollution in the Lesser district of Poland.

The authors of [19] proposed the first attempt to model mesh transformations by applying the concept of graph-grammar for regular two-dimensional triangular meshes with h-adaptation. The authors used quasi-context sensitive graph grammar. However, this approach generated hanging nodes, with all the difficulties associated with managing these nodes.

Another attempt utilized the Composite Programmable graph grammars (CP-graph grammar) introduced originally by [25–27] as a tool for a formal description of various design processes. The authors [21, 60–62] applied the CP-graph grammars to model two- and three-dimensional adaptive grids with hanging nodes.

In this thesis, we use the concept of a hypergraph, defined in Section 2.2.1. The hypergraphs and their grammars have been initially introduced by [32, 33] for some applications proposed in the field of computer graphics. There are also some special algorithms developed and optimized for the hypergraphs [36, 39, 56]. This work uses hyper-graphs to model the mesh refinement algorithm and work with the iterative GMRES solver. We have implemented our graph grammar-based system in the GALOIS framework [20, 34, 42, 44, 66], which allows for concurrent graph processing.

We use the Shuttle Radar Topography Mission [16] topography database to adaptively generate a two-diagonal triangular mesh representing Lesser district of Poland area. We use the longest edge refinements and remove hanging nodes. Finally, we extend the topographic mesh to a three-dimensional quadrilateral mesh representing the air above the terrain.

The resulting three-dimensional mesh is subjected to computer simulation with an advection-diffusion-time-difference solver modeling air pollution propagation. We apply the Streamline Upwind Petrov-Galerkin stabilization [37] for the advection-

diffusion-reaction problem. We use graph-grammar productions, generating element matrices and right-hand side vectors for each quadrilateral element. We include the Crank-Nicolson time integration scheme and interface with the iterative GMRES solver.

The motivations for developing a graph-grammar-based simulation system are as follows. We will compare the computational cost of the classical longest edge refinement algorithm [73] with our graph-grammar algorithm on a model example. We will count the number of basic operations, such as checking the state of a single triangle and breaking a single triangle. Although we do not know how to derive a formula for the computational cost for a general grid, we will compare the algorithms on a representative model example. The classical Rivara algorithm allows for parallelization by assigning each longest edge path to a single core. In our graph grammar-based algorithm, we go further and allow processing a single longest edge path with multiple cores.

We also argue that developing a graph-grammar-based system has some potential benefits for parallelizing computation. The productions of graph grammars are basic indivisible tasks that can be executed simultaneously. The graph-grammar model allows implementation in a graph-processing system such as the GALOIS environment [66]. Parallelism is obtained for free because the GALOIS system automatically manages the concurrent processing of graph-grammar productions. We implemented a graph-grammar-based model for grid generation and elementary matrix generation in this work. The element matrices and right-hand side vectors are then generated element-wise, and they are submitted to iterative solver without formulation of the global matrix. Iterative solver performs matrix-vector multiplications with respect to elements, multiplying elementary matrices by local parts of the right-hand side, without matrix assembling, but summing up the resulting vector.

1.2.2 *Longest edge refinement algorithm*

In this section, we present a description of the longest-edge mesh refinement algorithm introduced by Cecilia Rivara in 1984 [72]. The goal of the algorith is the bisection of simplices; be a set $q = \{a_1, a_2, \ldots, a_{n+1}\}$ of independent points in $\mathbb{R}^n$, Rivara defines the diameter of the simplex as

$$\delta(q) = \max\{\delta(\langle a_i, a_j \rangle) \forall i, j \in [1, n+1]\}$$

where $\delta(\langle a_i, a_j \rangle) = \|a_i - a_j\|_2$.

Therefore, there are two points a_k, a_m such that $\delta(q) = \delta(\langle a_k, a_m \rangle)$. Then, the bisection of the simplex q consists in adding a new point $a = (a_k + a_m)/2$, and spliting the simplex q in two new simplices q_k and q_m such that

$$q_k = \{a_1, a_2, \cdots, a_{k-1}, a, a_{k+1}, \cdots, a_m, \cdots, a_{n+1}\}$$
$$q_m = \{a_1, a_2, \cdots, a_k, \cdots, a_{m-1}, a, a_{m+1}, \cdots, a_{n+1}\}$$

The Rivara algorithm proposes a method to ensure the conformity of the mesh: the Longest-Edge Propagation Path (*LEPP*) [73]. The main idea is to partition an edge only when it is the longest-edge of all its adjacent simplices. Such the longest edge we call the terminal edge. When a simplex τ_0 is marked for refinement, we need to traverse all the adjacent elements edges until we find a terminal edge. All the traversal elements are called the Longest-Edge Propagation Path, and they constitute the set $LEPP(\tau_0)$. The algorithm partition the last two elements of the set $LEPP(\tau_0)$ and reconstructs the LEPP again until $LEPP(\tau_0)$ it is empty.

In 2009, Rivara published a review of the longest-edge bisecting method, [71], in which she describes the main properties of this method:

- The iterative and arbitrary use of this method produces triangles whose smallest interior angle is always greater than or equal to half of the smallest interior angle of the initial mesh. Moreover, there is a similarity between the generated triangles. This property proves the non-degeneracy of the algorithm.
- The division of the longest edge always ends with a finite number of steps.
- The relationship between the diameter of two adjacent triangles is positive and larger than a constant (K) that depends on the initial triangulation. This property ensures smoothness (no abrupt size change) of the new mesh.
- A global iterative application of the method to any triangulation generates most new triangles with the smallest angles greater than 30 degrees.

In 2013, Rivara proposed a parallel multi-threaded version of the *LEPP* algorithm [74]. This algorithm partition the problem by assigning different *LEPP* of different triangles marked to be refined to different threads.

1.2.3 *Modeling of finite element method with hanging nodes by graph grammar productions*

Alternative way of performing adaptive finite element method computations uses mesh refinements allowing for hanging nodes [12, 13, 31]. They use the 1-regularity rule, in which an element can be broken only once without breaking neighboring elements. The disadvantage of this method is the need to manage the forced approximation on broken edges with hanging nodes. The use of graph grammars to model mesh refinements with hanging nodes began with the work of [19] for regular triangular two-dimensional meshes with h-adaptation. In [80], refinements of the adaptive mesh with hanging nodes were modeled by a rewriting framework. There are several papers describing a graph-grammar approach for the hp-adaptive finite element method in which a 1-regularity rule breaks the mesh and the polynomial approximation rows of [15, 19, 24, 58, 59, 76, 78, 79] are changed. These models use complex graph grammars.

There are several applications designed for the definition and processing of graph grammars and their production (e.g. AGG, GROOVE, PROGRESS, VIATRA). These applications implement grammar transformation systems that are abstract encapsulations of several graph grammar models [4, 8–11, 41, 46]

1.2.4 *Air pollution dispersion modeling*

Air pollution is a critical topic and has a major impact on public health [57, 84], local environmental issues [1, 28], socio-economic impacts [35, 47] and also one of the main drivers of climate change [64, 68]. Thus, developing numerical methods for simulating the transport and response of air pollutants in the atmosphere is of great importance. However, it remains a scientific challenge to this day.

Air quality modeling systems mainly consist of three different components: the first component models emissions, the second component models the meteorological aspects of the computational domain, and the third model accounts for the chemical reactions between pollutant components.

This work focuses on adaptive finite element method simulations of air pollution phenomena using advection-diffusion-reaction models. The forces model the emission aspect on the right-hand side, the initial condition models the meteorological

aspects, the advection field modeling the wind, and the reaction component models the chemical aspect.

Some examples of applications of unstructured grids in pollution simulation problems are two-dimensional models [3, 43], and the three-dimensional models, including local refinement with element sizes of 2 km. presented in [82], and the three-dimensional tetrahedral meshes for local wind field analysis see [48–50, 52]. Several two-dimensional [85] and three-dimensional adaptive finite element method simulations have also been proposed [18, 51, 53]. Simulations of advection-diffusion-reaction models by the stabilized finite element method are described in [54, 55, 65] and [48, 49]. The chemical reactions are modeled using the RIVAD chemical model [77], or the CB05 chemical mechanism [86].

1.2.5 *Alternating direction solvers*

An alternative approach to reliable atmospheric simulations is to modify the alternating directions solver to work with irregular terrain geometry.

The alternating directions solver allows for a linear $O(N)$ factorization of the computational cost of matrices having a Kronecker product structure [14].

Let us consider a system of linear equations

$$\mathbf{M}\mathbf{x} = \mathbf{b}$$

with the matrix possessing a Kronecker product structure, namely $\mathbf{M} = \mathbf{A} \otimes \mathbf{B}$, where $\mathbf{A}$ is $n \times n$, $\mathbf{B}$ is $m \times m$. From the definition of the Kronecker product, we have

$$\mathbf{M} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A}\,B_{11} & \mathbf{A}\,B_{12} & \cdots & \mathbf{A}\,B_{1m} \\ \mathbf{A}\,B_{21} & \mathbf{A}\,B_{22} & \cdots & \mathbf{A}\,B_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}\,B_{m1} & \mathbf{A}\,B_{m2} & \cdots & \mathbf{A}\,B_{mm} \end{bmatrix}$$

We partition the unknowns vector and the right-hand side vector into m blocks. Each block has size n.

$$\mathbf{x}_i = (x_{i1}, \ldots, x_{in})^T$$
$$\mathbf{b}_i = (b_{i1}, \ldots, b_{in})^T$$

We can rewrite the system of linear equations in a block matrix form:

$$\begin{cases} \mathbf{A}B_{11}\mathbf{x}_1 + \mathbf{A}B_{12}\mathbf{x}_2 + \cdots + \mathbf{A}B_{1m}\mathbf{x}_m = \mathbf{b}_1 \\ \mathbf{A}B_{21}\mathbf{x}_1 + \mathbf{A}B_{22}\mathbf{x}_2 + \cdots + \mathbf{A}B_{2m}\mathbf{x}_m = \mathbf{b}_2 \\ \quad + \quad + \quad + \quad = \\ \mathbf{A}B_{m1}\mathbf{x}_1 + \mathbf{A}B_{m2}\mathbf{x}_2 + \cdots + \mathbf{A}B_{mm}\mathbf{x}_m = \mathbf{b}_m \end{cases}$$

We can factor out $\mathbf{A}$ in this blocked system:

$$\begin{cases} \mathbf{A}\left(B_{11}\mathbf{x}_1 + B_{12}\mathbf{x}_2 + \cdots + B_{1m}\mathbf{x}_m\right) = \mathbf{b}_1 \\ \mathbf{A}\left(B_{21}\mathbf{x}_1 + B_{22}\mathbf{x}_2 + \cdots + B_{2m}\mathbf{x}_m\right) = \mathbf{b}_2 \\ \quad + \quad + \quad + \quad = \\ \mathbf{A}\left(B_{m1}\mathbf{x}_1 + B_{m2}\mathbf{x}_2 + \cdots + B_{mm}\mathbf{x}_m\right) = \mathbf{b}_m \end{cases}$$

We multiply the blocked system by $\mathbf{A}^{-1}$. We define $\mathbf{y}^i = \mathbf{A}^{-1}\mathbf{b}^i$. After this operation we obtain the first one-dimensional system $\mathbf{A}\,\mathbf{y}^i = \mathbf{b}^i$ with multiple right-hand-sides.

$$\begin{cases} B_{11}\mathbf{x}_1 + B_{12}\mathbf{x}_2 + \cdots + B_{1m}\mathbf{x}_m = \mathbf{y}_1 \\ B_{21}\mathbf{x}_1 + B_{22}\mathbf{x}_2 + \cdots + B_{2m}\mathbf{x}_m = \mathbf{y}_2 \\ \quad + \quad + \quad + \quad = \\ B_{m1}\mathbf{x}_1 + B_{m2}\mathbf{x}_2 + \cdots + B_{mm}\mathbf{x}_m = \mathbf{y}_m \end{cases}$$

We consider now each component of vectors $\mathbf{x}_i$ and $\mathbf{y}_i$, and for each $i = 1, ..., n$ we obtain a linear system. Thus, we have a family of linear systems of equations

$$\begin{cases} B_{11}x^{1i} + B_{12}x^{2i} + \cdots + B_{1m}x^{mi} = y_{1i} \\ B_{21}x^{1i} + B_{22}x^{2i} + \cdots + B_{2m}x^{mi} = y_{2i} \\ \quad + \quad + \quad + \quad = \\ B_{m1}x^{1i} + B_{m2}x^{2i} + \cdots + B_{mm}x^{mi} = y_{mi} \end{cases}$$

for each $i = 1, \ldots, n$. Each of these linear systems of equations has identical matrix $\mathbf{B}$. Here we have just obtained the second one-dimensional system of linear equations with multiple right-hand-sides $\mathbf{B}\,\mathbf{x}^i = \mathbf{y}^i$.

Alternating-direction methods modify the original linear systems of equations in an effort to reduce computational cost [81]. The domain decomposition methods divide the resulting system into sub-operators, which divide the vector of unknowns into smaller subsets [22, 69]). However, the splitting methods

require method-specific stability and error analysis (see, [45, 83] for more details).

The alternating-direction solver uses regular three-dimensional computational grids and does not allow grid adaptation by default. However, it allows solving complex systems of linear equations at linear computational cost using special time integration schemes that result in a Kronecker matrix structure.

1.3 MAIN SCIENTIFIC CONTRIBUTIONS

The new contributions of this thesis can be summarized as follows:

- we focus on the algorithm allowing for pollution simulations over complicated terrain topography, with adaptive finite element method and graph grammar based longest-edge refinement method using advection-diffusion-reaction model
- the parallelism in the original Rivara's longest edge refinement algorithm is achieved by processing different longest edge refinement paths in different cores, while our graph-grammar-based algorithm allows for additional parallelization within a single longest-edge refinement path,
- we express the Rivara algorithm by graph-grammar productions, and use it for the topographic mesh generation,
- we extend it to model the generation of the three-dimensional tetrahedral meshes span over the terrain mesh,
- we express the stabilized finite element method of the non-stationary advection-diffusion-reaction simulator by graph grammar productions, working on top of the three-dimensional tetrahedral finite element mesh,
- we incorporate the time-integration scheme of Crank-Nicolson, and we interface with the iterative GMRES solver, performing the matrix-vector multiplications element wise, without formulating the global matrix.
- we perform the pollution simulations in Lesser Poland area.
- we focus the alternating direction solver with finite difference method over regular computational domain using

Navier-Stokes-Boussinesq solver and we propose a modification of the alternating direction solver allowing for incorporation of the non-regular terrain geometry preserving linear computational cost of the solver

- We verify the computational cost of the alternating direction solver on parallel Linux cluster

Part II

METHODS AND ALGORITHMS

2

METHODS AND ALGORITHMS

2.1 KEY CONCEPTS

We start with introduction of the concept of hypergraph and graph grammar.

Definition 1. An undirected attributed labeled hypergraph over label alphabet C and attribute set A is defined as a system $G = (V, HE, t, l, at, val)$, where:

- V is a finite set of nodes,
- HE is a finite set of hyperedges,
- $t : HE \to V^*$ is a mapping assigning sequences of target nodes to hyperedges,
- $l : V \cup HE \to C$ is a node and hyperedge labeling function,
- $at : V \cup HE \to 2^A$ is a node and hyperedge attributing function, where 2^A is a power set of A.
- $val : (V \cup HE) \times A \to D$ is a function assigning values of attributes of nodes and hyperedges, where $D = \bigcup_{a \in A} D_a$ where D_a is a set of admissible values of attribute a.

Definition 2. A hypergraph $G_2 = (V_2, HE_2, t, l, at, val_1)$ over C and A is a subgraph of a hypergraph $G_1 = (V_1, HE_1, t, l, at, val_2)$ over C and A, (i.e., $G_2 \subseteq G_1$) if $V_2 \subseteq V_1$ and $E_2 \subseteq E_1$.

Definition 3. A production suitable hypergraph of type k is a system $H = (G, Ext)$, where:

- $G = (V, HE, t, l, at, val)$ is a hypergraph over C and A,
- Ext is a sequence of external nodes of V, with $|Ext| = k$, where $(Ext = (Ext_1, Ext_2, \ldots, Ext_k))$.

Remark 1. Let G_2 be a subgraph of If we need G_2 to be a production suitable hypergraph $H_2 = (G_2, Ext_2)$ then we need to define Ext_2 in the following way. Let $G_3 = (V_3, HE_3, t, l, at, val_3)$ where $HE_3 = HE_1 - HE_2$ and V_3 are the nodes in V_1 that are connected to HE_3. Then, the nodes in Ext_2 are $V_3 \cap V_2$; that is, the nodes that connect H_2 and H_3.

Definition 4. A hypergraph production is a pair $p = (L, R)$, where both L and R are production suitable hypergraphs of the same type k (both having the same number of external nodes k).

Two graphs are isomorphic if they both have the same number of nodes and edges. The corresponding nodes of both graphs have the same labels and attributes. The corresponding edges of both graphs have the same labels, attributes, and sequences of nodes belonging to them.

Definition 5. Graph G is isomorphic up to attribute values with graph G' iff there exist bijections $f : V \to V'$ and $g : EH \to EH'$ such that

- $(l(v) = l'(f(v)))\ \forall v \in V$,
- $(l(e) = l'(g(e)))\ \forall e \in EH$,
- $(at(v) = at'(f(v)))\ \forall v \in V$,
- $(at(e) = at'(g(e)))\ \forall e \in EH$,
- $(v_1, v_2, \ldots, v_k) = t(e)$ iff $(f(v_1), f(v_2), \ldots, f(v_k)) = t'(g(e))$ $\forall e \in E, \forall v_1 \in V, v_2 \in V, \ldots, v_k \in V$.

The application of the graph-grammar production $p = (L, R)$, where L is isomorphic with the hypergraph $H_2 = (G_2, Ext_2)$ and R is isomorphic with the hypergraph $H_4 = (G_4, Ext_4)$ to the hypergraph G_1 consists in removing the production suitable hypergraph H_2 from G_1, replacing it by the production suitable hypergraph H_4, and connecting external nodes of H_4 with the hyperedges of the hypergraph $G_1 \backslash G_2$ in such a way that each hyperedge which connected the node v of $G_1 \backslash G_2$ with the external node Ext_{2_i} of H_2 before application of the production, where $i = 1, \ldots, k$, now connects node v of $G_1 \backslash G_2$ with the external node Ext_{4_i} of H_4. As the result, the graph G_1 is transformed into G_5, where the set of nodes is equal to $V_1 \backslash V_2 \cup V_4$ and the set of edges is equal to $EH_1 \backslash EH_2 \cup EH_4$.

The definition of graph grammar creation can be extended by adding a condition over the labels and attribute values of hypergraphs, called the applicability predicate, which determines whether hypergraph creation can be applied.

Definition 6. A hypergraph production with applicability predicate is a triple $p = (L, R, r)$, where both L and R are production suitable hypergraphs of the same type and r is applicability predicate defined as: $r : \mathcal{F} \rightarrow \{TRUE, FALSE\}$, where $\mathcal{F}$ is a set of logical expressions defined over labels and values of attributes of the hypergraphs.

Production with an applicability predicate may be applied to a graph only in the case that the applicability predicate is satisfied.

Definition 7. A hypergraph grammar is a system $GG = (P, G_S)$, where:

- P is a finite set of hypergraph productions of the form $p = (L, R, r)$.
- G_S is an initial hypergraph.

Definition 8. Let GG be a hypergraph grammar, and Ω be a set of labels of productions from GG. A control diagram CD over Ω is a directed labeled graph $CD = ((V, E), \xi_V, I, F)$ where

- V is a set of graph nodes, E is a set of directed graph edges.
- There exists exactly one node labeled I and exactly one node labeled F.
- There is no edge in-comming to the initial node labeled by I, and there is no edge out-comming from the final node labeled by F, and the other nodes has at least one in-comming and one out-comming edge.
- $\xi_V : V \rightarrow \Omega \cup \{I, F\}$, is a node labeling function.

The control diagram defines the execution and planning strategy of the hypergraph grammar. We start at the initial node I and follow a directed graph, checking applicability predicates and applying the production identified by the label from each control diagram node to the hypergraph.

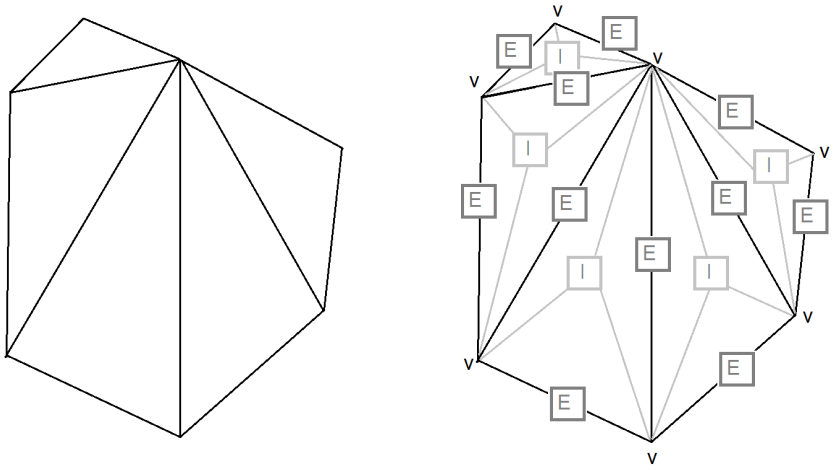


Figure 1: Hipergraph representation of the computational mesh.

2.2 GRAPH-GRAMMAR-BASED LONGEST EDGE REFINEMENT ALGORITHM IN TWO DIMENSIONS

In this section, we present a two-dimensional mesh refinement algorithm initially proposed by Rivara [73]. We express the algorithm for the first time using the graph grammars. Instead of following the idea of the Longest-Edge Propagation Path algorithm, we will define a hypergraph that models an unstructured triangular mesh and a set of productions that locally modify the triangles.

We represent the computational mesh as a hipergraph with graph vertices **v** representing mesh nodes, graph edges **E** representing mesh edges, and graph hyperedges **I** representing element interiors.

The productions are set so that all of them bisect a triangle, so they must be applied only to triangles marked for refinement or triangles that must be split in half to conform to the mesh, i.e., triangles with one, two, or three hanging nodes.

Remark 2. The criterion for choosing the longest-edge is to prevent propagation of the unwanted additional refinements. To fullfil this criterion, we setup the priority for edges refinements: First, we break edges with hanging-nodes; Second, we break edges on the boundary; Third, we break the remaining edges.

We illustrate the concept of the graph-grammar-based longest edge refinement on the following example, presented in Figure 2. In this example, we want to break one triangle located at the top of the mesh. This requires the following sequence of productions **(**P_1**)-(**P_3**)-(**P_3**)-(**P_2**)-(**P_2**)-(**P_3**)-(**P_2**)-(**P_3**)-(**P_2**)-(**P_3**)-(**P_2**)-(**P_2**)**.

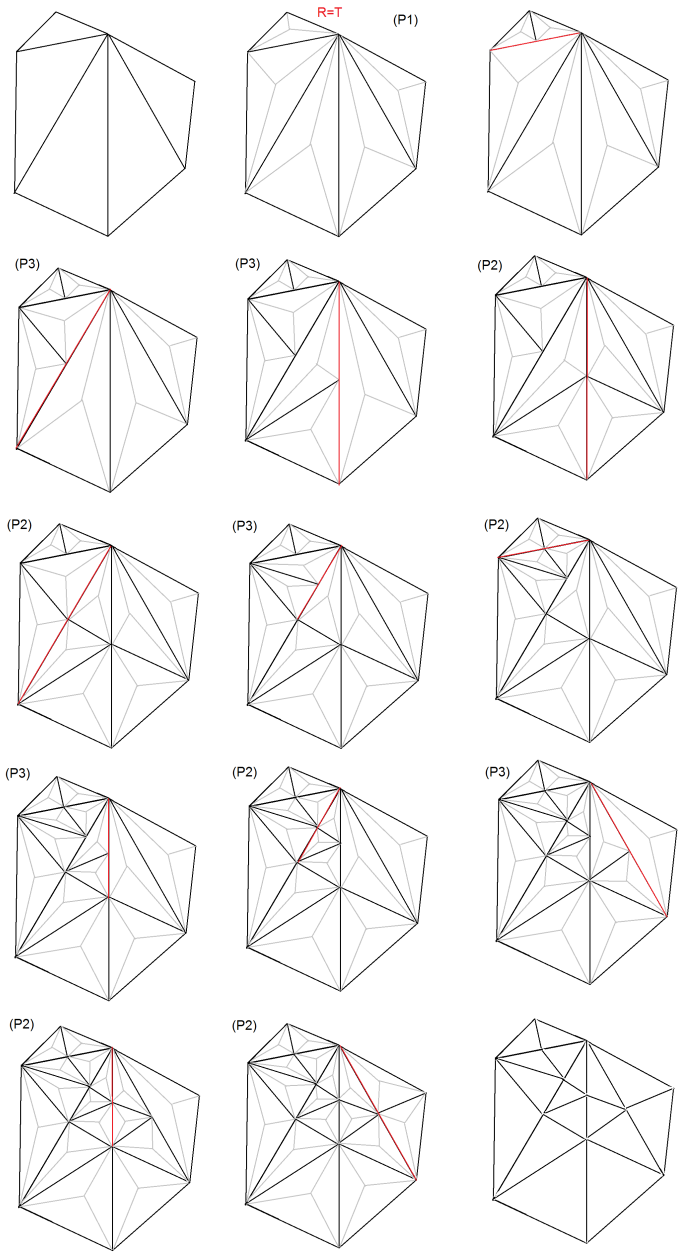


Figure 2: Exemplary derivation. By black lines we denote triangles, by light gray lines we denote the element interior hyperedges, by red lines we denote the longest edges where the production is applied. We print the productions names when it is applied. The first plot denotes the starting mesh. The triangle to be broken is denoted be **R=T**. The final plot denotes the final mesh.

We represent the computational mesh by a hypergraph. Black lines denote triangles of the mesh, the light lines represent the interiors of element represented by hyperedges. The red line denotes the longest-edge of the considered sub-graph (representing a single mesh element). We want to refine top left corner element. We setup refinement flag R=T. Production **(*P1*)** breaks the element towards the longest edge, the edge is labeled as having a hanging node. Production **(*P3*)** breaks the element with an edge having a hanging node, and it generates another edge with hanging node. Production **(*P2*)** breaks the element with an edge having a hanging node, and it does not produce new hanging nodes. The refinements propagates through the longest-edge refinement flags, as executed by production **(*P3*)** until they reach the terminal edge, as expressed by production **(*P2*)**. The process of formation of longest-edge refinement path by productions **(*P3*)** and is closure by production **(*P2*)** is repeated several times until there are no hanging nodes present in the mesh, and all the element set to be refined have been broken.

Below we provide the formal definitions of the hypergraph-grammar productions.

2.2.1 *Hypergraph definition*

The hypergraph modelling an unstructured mesh with triangular elements is defined with the set of labels $C = \{N, E, T\}$ and attributes $A = \{x, y, z, HN, B, L, R\}$, where

- N is a hypergraph node label that represents a triangular element node.
- E is a hyperedge label that denotes an edge of a triangular element.
- T is a hyperedge label that denotes an interior of a triangular element.
- x is a hypergraph node attribute which denotes x coordinate of the node, where $D_x \subset \mathbb{R}$.
- y is a hypergraph node attribute which denotes y coordinate of the node, where $D_y \subset \mathbb{R}$.
- z is a hypergraph node attribute which denotes z coordinate of the node, where $D_z \subset \mathbb{R}$.

- *HN* is a hypergraph node attribute which denotes if the corresponding triangular element node is a hanging node, where $D_{HN} = \{TRUE, FALSE\}$.
- B is a hyperedge attribute which denotes if the corresponding triangular element edge is located on the boundary of the triangular mesh, where $D_B = \{TRUE, FALSE\}$.
- L is a hyperedge attribute that denotes the corresponding triangular element edge's length, where $D_L \subset \mathbb{R}$.
- R is a hyperedge attribute which denotes if the corresponding triangular element is to be refined, where $D_R = \{TRUE, FALSE\}$.

2.2.2 *Productions*

We introduce six graph-grammar productions to implement the longest-edge refinement algorithm. These productions can be summarized in the following way:

(p1) The triangle to be refined has no hanging-node and is marked to be refined. Predicate of applicability prioritizes the longest-edge on the border.

(p2) The triangle to be refined has one hanging-node; the longest-edge is the one that contains the hanging-node.

(p3) The triangle to be refined has one hanging-node; the longest-edge is one that does not contain the hanging-node. Predicate of applicability prioritizes the longest-edge on the border.

(p4) The triangle to be refined has two hanging-nodes; the longest-edge is one that contains a hanging-node.

(p5) The triangle to be refined has two hanging-nodes; the longest-edge is the one that does not contain a hanging-node. Predicate of applicability prioritizes the longest-edge on the border.

(p6) The triangle to be refined has three hanging-nodes.

Remark 3. Six productions assume that each edge of a triangle can contain only one hanging node. To ensure this restriction, the productions allow only one edge that is connected to two regular nodes (no hanging nodes) to be broken.

To improve the definitions, a new function (*NL*) was introduced. This function calculates the length of the new edge depending on the nodes. Depending on the number of arguments, there are two possibilities. So we have the following two functions:

$$NL(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$NL(i,j,k) = \sqrt{(\frac{x_i + x_j}{2} - x_k)^2 + (\frac{y_i + y_j}{2} - y_k)^2 + (\frac{z_i + z_j}{2} - z_k)^2}$$

In the following part of the section, we introduce the six graph-grammar productions and their predicates of applicability.

2.2.2.1 *Production 1* **(P1)**

Production **(P1)**, Fig. 3, describes the breaking of a triangular element with no hanging-nodes. This production will break the element by breaking edge 1.

The predicate of applicability is the following:

(*R1 AND* ((*L1* $\geqslant$ *L2*) *AND* (*L1* $\geqslant$ *L3*))). The first condition checks is the triangle is marked for refinement (*R1*), and the edge 1 is one the longest-edge ((*L1* $\geqslant$ *L2*) *AND* (*L1* $\geqslant$ *L3*)). Then, there are two cases:

- (*B1*) If edge 1 is located on the boundary of the domain (*B1*), then the element is broken. We prioritize breaking the boundary.
- (*NOT* B1 *AND* (*NOT HN1 AND NOT HN2*) *AND* (*NOT* ((B2 *AND L2 = L1*) OR (*B3 AND L3 = L1*)))) If edge 1 is not on the boundary (*NOT* B1), we need to ensure that the two nodes of edge 1 are not hanging nodes (*NOT HN1 AND NOT HN2*); if they are, we cannot break the edge 1. We also need to make sure that there is no other longest-edge on the boundary (*NOT* ((B2 *AND L2 = L1*) OR (*B3 AND L3 = L1*))).

This production breaks the longest edge. It creates a new hanging-node node at $(x = (x1 + x2)/2, y = (y1 + y2)/2, z = (z1 + z2)/2)$. If the edge is on the boundary it won't be a hanging node (*HN==B1*). The breaking of the edge also generates two new edges whose lengths are half the length of the broken edge (L=*L1*/2). They inherit the boundary flag (B=*B1*). Then the

triangle must be partitioned; to do so, it generates a new edge that connects the newly created node to the opposite node, calculating the length of this new edge (L=*NL*(1,2,3)) and setting it as an interior edge (B=*FALSE*). Finally, it generates the new two triangles with (R=*FALSE*).

2.2.2.2 *Production 2* **(P2)**

Production **(P2)**, Fig. 4 describes breaking of a triangle with one hanging-nodes by the edge that contains the hanging-node.

The predicate of applicability is the following:

$((L4+L5) \geqslant L2)$ *AND* $((L4+L5) \geqslant L3)$. The only condition in this production ensures that the broken edge (edge 4 + edge 5) is the longest-edge ($((L4+L5) \geqslant L2)$ *AND* $((L4+L5) \geqslant L3)$). We do not need to check that the triangle is marked for refinement because this dissection is needed for conformity. No other conditions are required because the edge with the hanging node has a higher priority.

This production does not break the longest edge because it is already broken. The production cuts the triangle in half; to do so, it generates a new edge that connects the newly created node to the opposite node, computing the length of this new edge (L=*NL*(4,3)) and setting it as an interior edge (B=*FALSE*). And finally, it generates the new two triangles that are not marked to be refined (R=*FALSE*).

2.2.2.3 *Production 3* **(P3)**

Production **(P3)**, Fig. 5 defines breaking of a triangle with one hanging-node by one edge that does not contain the hanging-node. This production will bisect the triangle by edge 3.

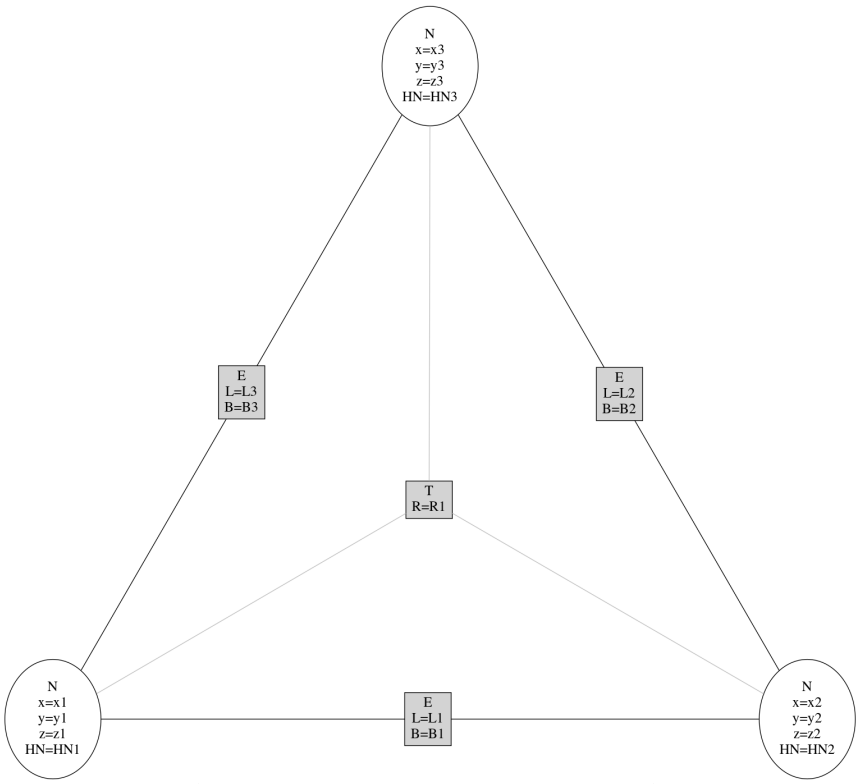
The predicate of applicability is the following:

($(L3 \geqslant L2)$ *AND* $(L3 > (L4+L5))$). The first condition ensures that edge 3 is longer or equal to edge 2 ($L3 \geqslant L2$). It ensures that edge 3 is strictly longer than the edge to be broken ($L3 > (L4+L5)$). If they would have the same length, the broken edge would have higher priority.

We don't need to check if the triangle is marked for refinement since this bisection is needed for conformity.

Once we know that edge 3 is suitable for breaking, there are two cases:

- ($B3$) If edge 3 is on the boundary ($B3$), then the triangle will be broken. We prioritize breaking the boundary. Also,



$$
\begin{aligned}
&R(R_1,B_1,B_2,B_3,L_1,L_2,L_3,HN_1,HN_2) = \\
&(R_1 \text{ AND } ((L_1 \geqslant L_2) \text{ AND } (L_1 \geqslant L_3)) \\
&\text{AND} \\
&[\ (B_1) \\
&\text{OR} \\
&(\ (\text{NOT } B_1) \text{ AND } (\text{NOT } HN_1 \text{ AND NOT } HN_2) \text{ AND} \\
&(\text{NOT } ((\ B_2 \text{ AND } L_2 = L_1) \text{ OR } (B_3 \text{ AND } L_3 = L_1)))\ )\ ]
\end{aligned}
$$

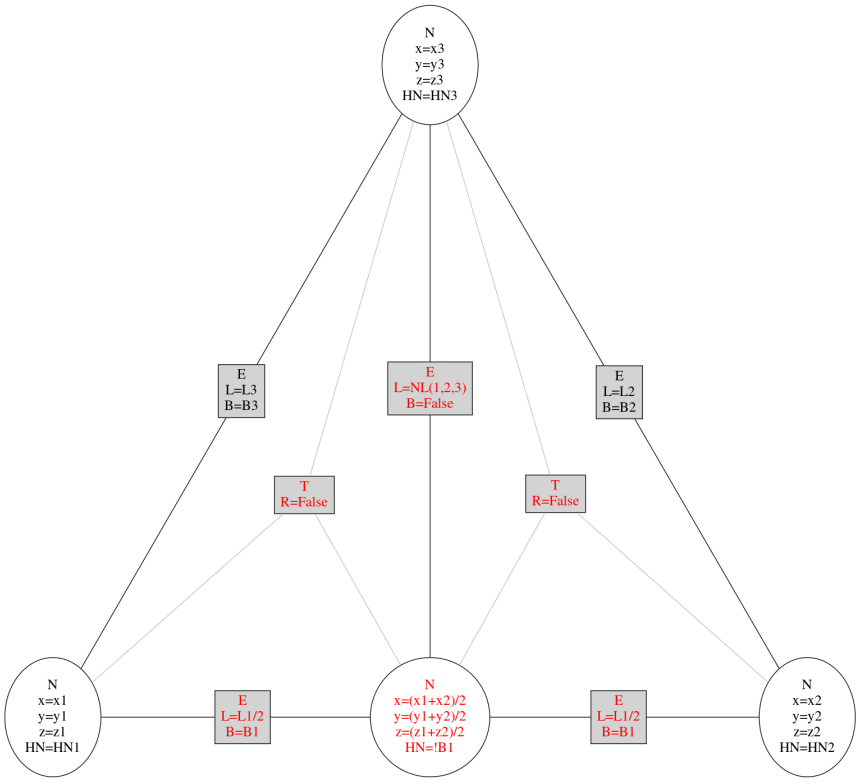


Figure 3: Production **(P1)** for the refinement of the marked element.

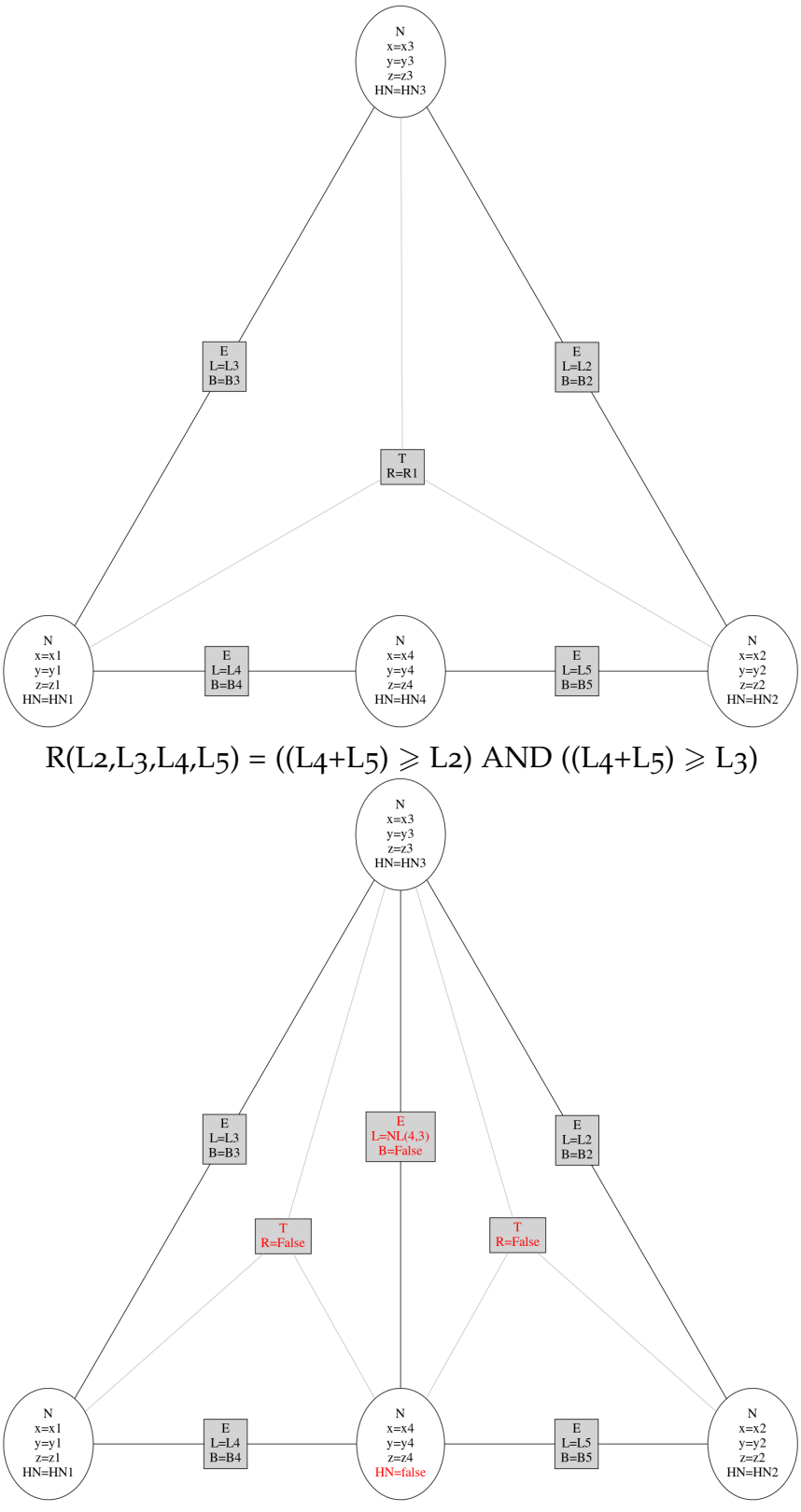


Figure 4: Production **(P2)** for the additional refinement of the element with the longest edge already broken with the hanging node, replacing the hanging node with the regular node.

there are any hanging nodes at the end of this edge on the boundary.

- ((*NOT B3*) *AND* (*NOT HN1 AND NOT HN3*) *AND* (*NOT* (B2 *AND L2 = L3*))) If edge 3 is not on the boundary (*NOT B3*), we need to ensure that the two nodes of edge 3 are not hanging nodes (*NOT HN1 AND NOT HN3*); if they are, we cannot break edge 3 yet. Finally, we should ensure that edge 2 is not a longest-edge on the boundary (*NOT* (B2 *AND L2 = L3*)).

This production breaks edge 3, generating a new hanging node at $(x = (x1 + x3)/2, y = (y1 + y3)/2, z = (z1 + z3)/2)$. If the edge is on the boundary, it won't be a hanging node (*HN=-B3*). The breaking of the edge also generates two new edges whose lengths are half the length of the broken edge (L=*L3*/2). They inherit the boundary flag (B=*B3*). Then, the triangle is broken. It generates a new edge that connects the newly created node with the opposite node. It computes the length of this new edge (L=*NL*(1,3,2)) and sets the interior edge flag (B=*FALSE*). It also generates two triangles that are not marked to be refined (R=*FALSE*).

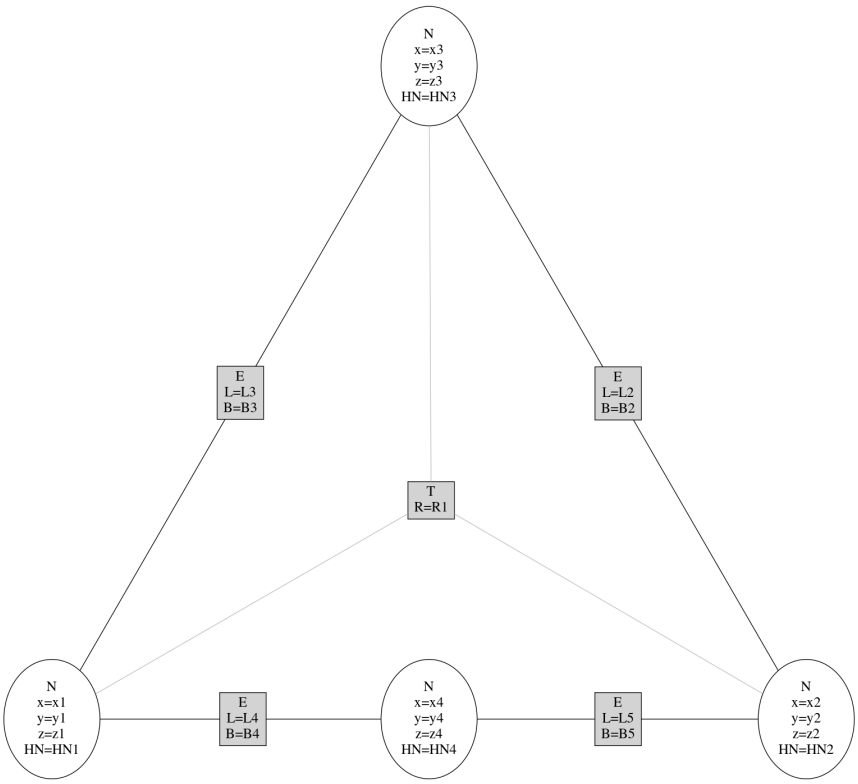
2.2.2.4 *Production 4* **(P4)**

Production **(P4)**, Fig. 6 defines breaking of a triangle with two hanging-nodes by one of the edges that contain a hanging-node. This production will break the triangle by the node connected to edge 4 and edge 5.

The predicate of applicability is the following:

(((*L4+L5*) $\geqslant$ (*L6+L7*)) *AND* ((*L4+L5*) $\geqslant$ *L3*)). The only condition in this production ensures that the broken edge (edge 4 + edge 5) is the longest-edges (((*L4+L5*) $\geqslant$ (*L6+L7*)) *AND* ((*L4+L5*) $\geqslant$ *L3*)). We don't check if the triangle is marked for refinement since this bisection is needed for conformity. No other conditions are required because the edge with the hanging node has a higher priority.

This production does not break the longest edge because it is already broken. The production cuts the triangle in half; to do so, it generates a new edge that connects the newly created node to the opposite node, computing the length of this new edge (L=*NL*(4,3)) and setting it as an interior edge (B=*FALSE*). And finally, it generates the two new triangles. They have the refinement flag unset (R=*FALSE*).



$$
\begin{gathered}
R(B2,B3,L2,L3,L4,L5,HN1,HN3) = \\
((L3 > (L4+L5)) \text{ AND } (L3 \geqslant L2)) \\
\text{AND} \\
[(B3) \\
\text{OR} \\
((\text{NOT } B3) \text{ AND } (\text{NOT } HN1 \text{ AND NOT } HN3) \text{ AND} \\
(\text{NOT } (B2 \text{ AND } L2 = L3)))]
\end{gathered}
$$

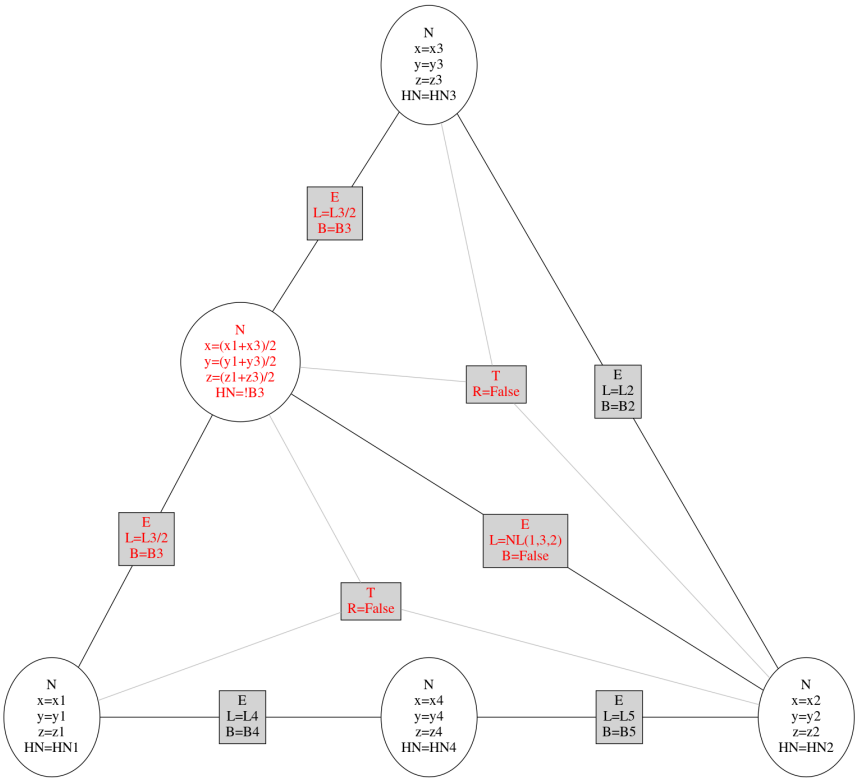
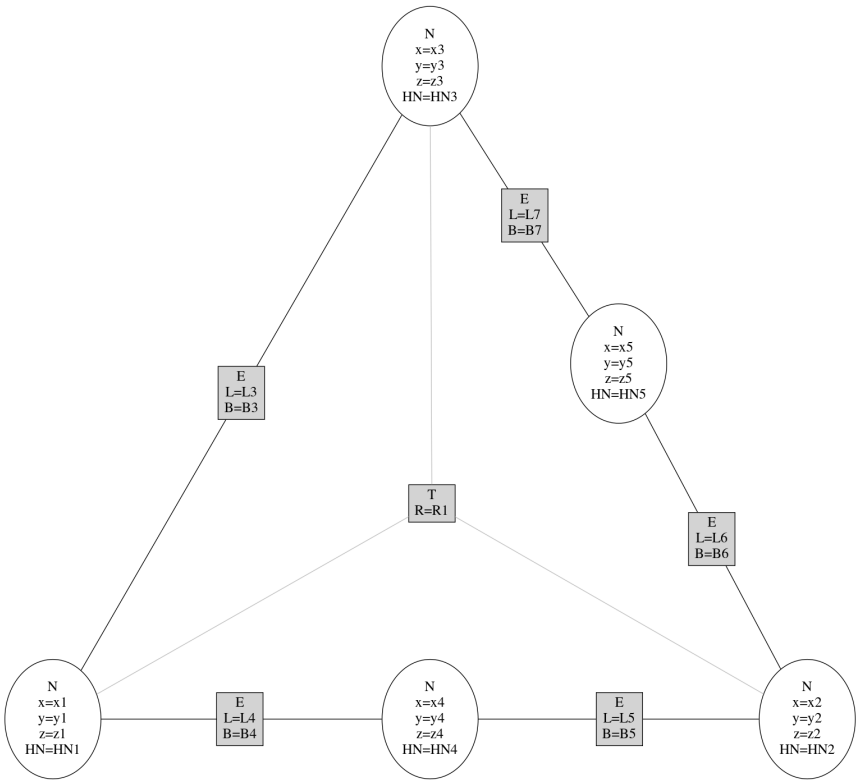


Figure 5: Production **(P3)** for the additional refinements of the element with one hanging node.



$$R(L3,L4,L5,L6,L7) = ((L4+L5) \geqslant (L6+L7)) \text{ AND } ((L4+L5) \geqslant L3)$$

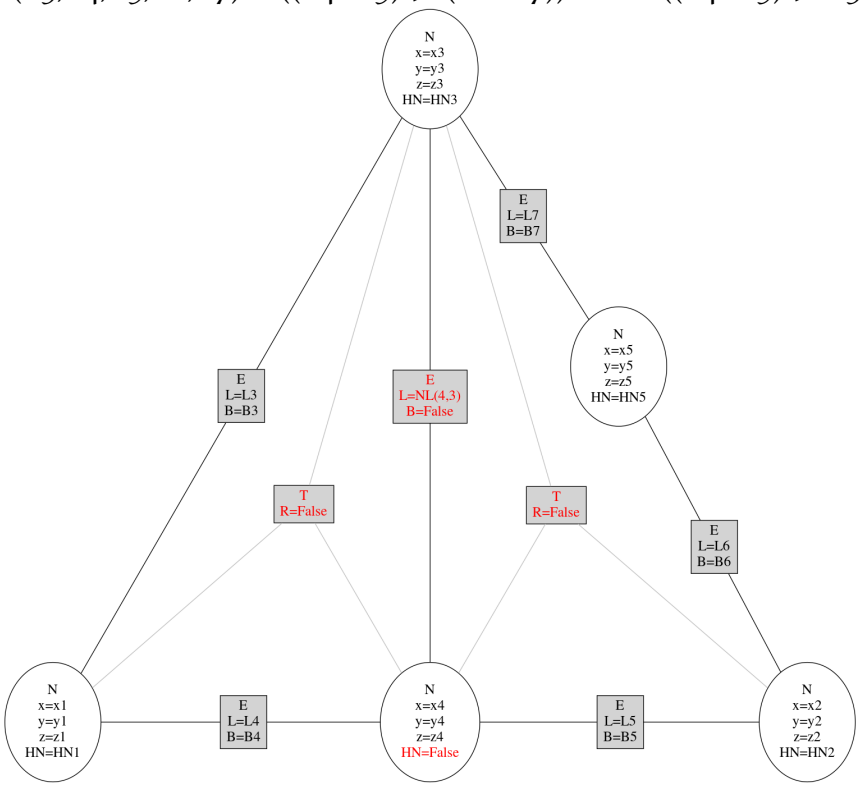


Figure 6: Production **(P4)** for an additional refinement of an element with two hanging nodes, breaking the element towards one of the broken edges

2.2.2.5 *Production 5* **(P5)**

Production **(P5)**, Fig. 7 defines breaking of a triangle with two hanging-nodes by the edge that does not contain a hanging-node. This production will break the triangle by edge 3.

The predicate of applicability is the following:

((($L3 > (L4+L5)$) *AND* ($L3 > (L6+L7)$)) *AND* (*NOT HN1 AND NOT HN3*)). The first condition in this production ensures that the edge 3 is the longest-edge (($L3 > (L4+L5)$) *AND* ($L3 > (L6+L7)$)); both comparisons are strictly greater because broken edges have priority. The second condition ensures that the two nodes of edge 3 are not hanging nodes (*NOT HN1 AND NOT HN3*). Otherwise, we cannot break edge. We don't need to check if the triangle is marked to be refined since this bisection is needed for conformity.

This production breaks edge 3. It generates a new node at $(x = (x1 + x3)/2, y = (y1 + y3)/2, z = (z1 + z3)/2)$. This new node won't be a hanging if the edge is on the boundary (*HN*==$B3$). The breaking of the edge also generates two new edges whose lengths are half the length of the broken edge (L=$L3/2$). They inherit the boundary flag (B=$B3$). Then, the triangle is broken. It generates a new edge that connects the newly created node with the opposite node, computing the length of this new edge (L=*NL*(1,3,2)) and setting it as an interior edge (B=*FALSE*). It also generates the two new triangles, with the flags (R=*FALSE*).

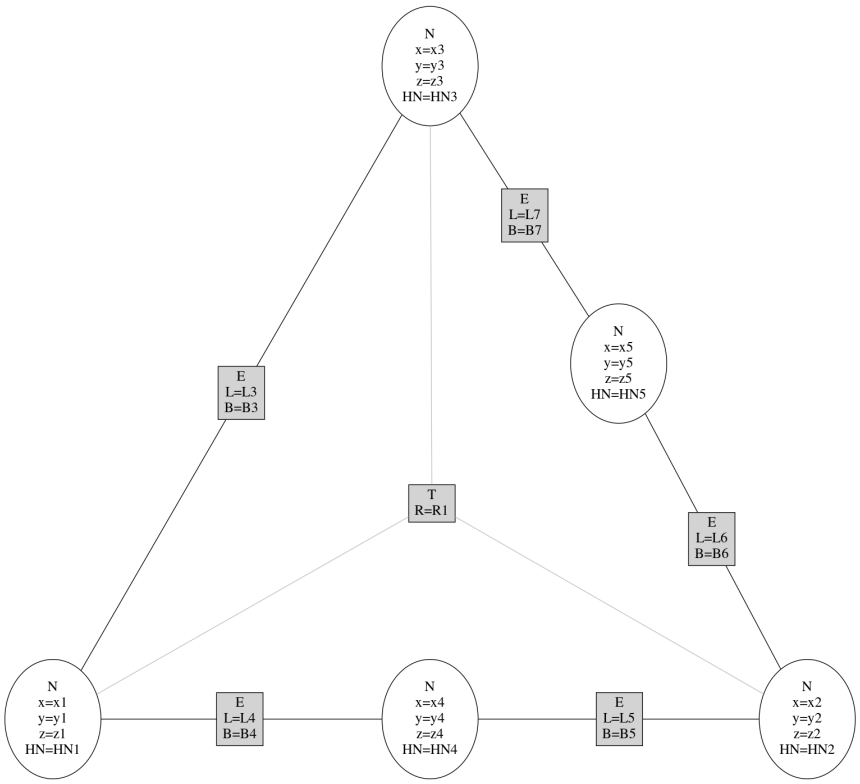
2.2.2.6 *Production 6* **(P6)**

Production **(P6)**, Fig. 8 defines breaking of a triangle with three hanging-nodes. This production will break the triangle by the node connected to edge 4 and edge 5.

The predicate of applicability is the following:

((($L4+L5$) $\geqslant$ ($L6+L7$)) *AND* (($L4+L5$) $\geqslant$ ($L8+L9$))). The only condition in this production ensures that the broken edge (edge 4 + edge 5) is the longest-edge ((($L4+L5$) $\geqslant$ ($L6+L7$)) *AND* (($L4+L5$) $\geqslant$ ($L8+L9$))). We don't need to check if the triangle is marked to be refined since this bisection is needed for conformity. No other conditions are required since an edge with a hanging-node has the higher priority.

This production does not break the longest edge. The longest edge is already broken. It generates a new edge that connects the newly created node with the opposite node, computing the length of this new edge (L=*NL*(4,3)) and setting it as an interior



$$R(B_3,L_3,L_4,L_5,L_6,L_7) = ((L_3 > (L_4+L_5)) \text{ AND } (L_3 > (L_6+L_7))) \text{ AND } (\text{NOT } HN_1 \text{ AND NOT } HN_3)$$

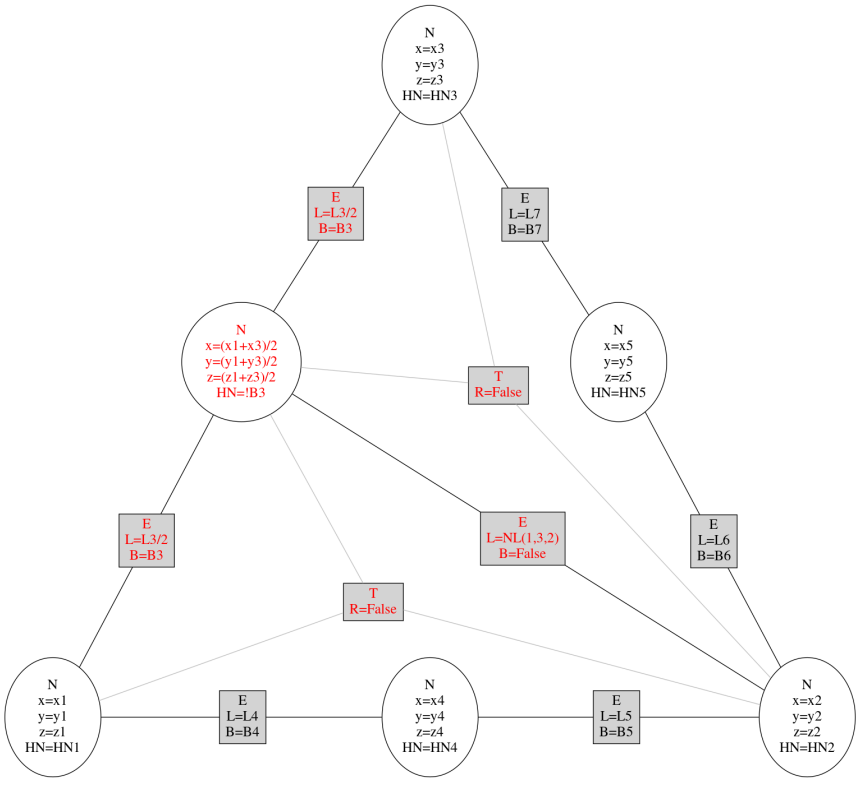


Figure 7: Production **(P5)** for an additional refinement of the element with two hanging nodes, breaking the element towards the unbroken edge.

edge (B=*FALSE*). And finally, it generates the new two triangles. They have the flags (R=*FALSE*).

2.2.3 *Control diagram*

A control diagram of the execution of graph grammar productions is shown in Fig. 9. In a given state, we try to execute productions from that state in all possible locations of the hypergraph representing the grid. The transition to another state occurs when we have performed production at all possible locations. We loop until the production can no longer be executed.

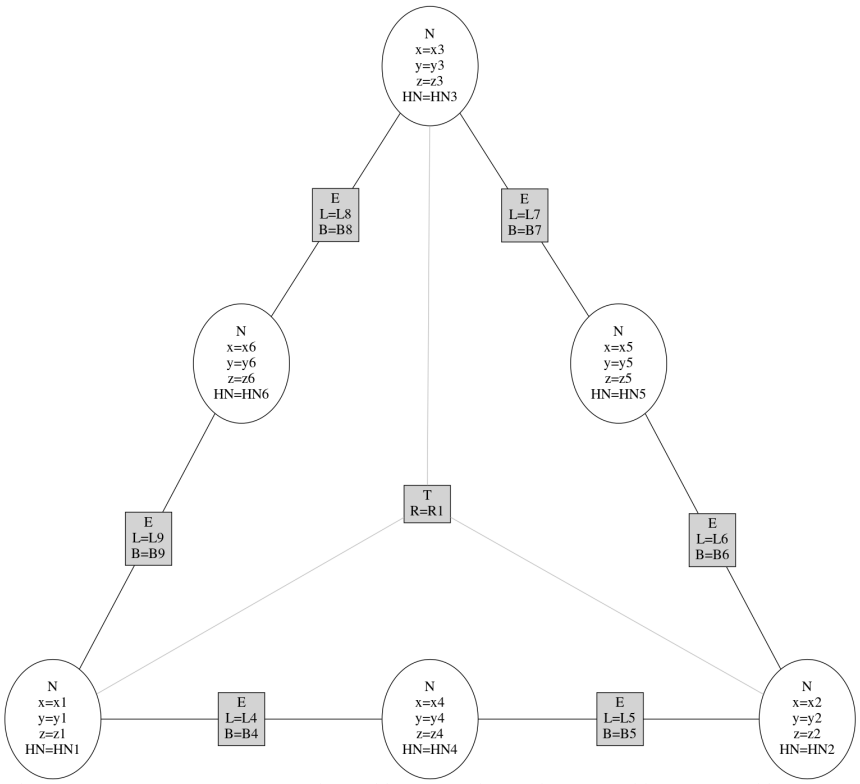
2.3 GALOIS IMPLEMENTATION OF THE LONGEST-EDGE REFINEMENT ALGORITHM

We implemented our graph grammar-based system in the GALOIS framework [20, 66], which enables concurrent graph processing. The GALOIS implementation of the graph-grammar-based mesh generator is stored at *https://github.com/ Podsiadlo/ TerrainMeshGenerator/ tree/ graphgrammar/lonestar/ graphgrammar2*

Below we provide some snapshots on the source code.

The main routine contains parallel loop over all triangles, and it runs the error estimator `checker.execute(node)` over each triangle to decide if it has to be broken. A triangle is represented by its internal hyperedge, implemented in Galois as a graph node `node`. Next, it loops through all the triangles again, then it loops over all six productions, and it tries to execute each of the six production over each element by calling `production-execute(pState, ctx)`.

```
for (int j = 0; j <config.steps; j++) {
  galois::for_each(galois::iterate(graph.begin(), graph.end()),
                   [&](GNode node, auto &ctx) {
    if (basicCondition(graph, node)) {
      checker.execute(node);
    }
  });
  galois::for_each(galois::iterate(graph.begin(), graph.end()),
                   [&](GNode node, auto &ctx) {
    if (!basicCondition(graph, node)) {
      return;
    }
```



$$R(L3,L4,L5,L6,L7,L8,L9) = ((L4+L5) \geqslant (L6+L7)) \text{ AND } ((L4+L5) \geqslant (L8+L9))$$

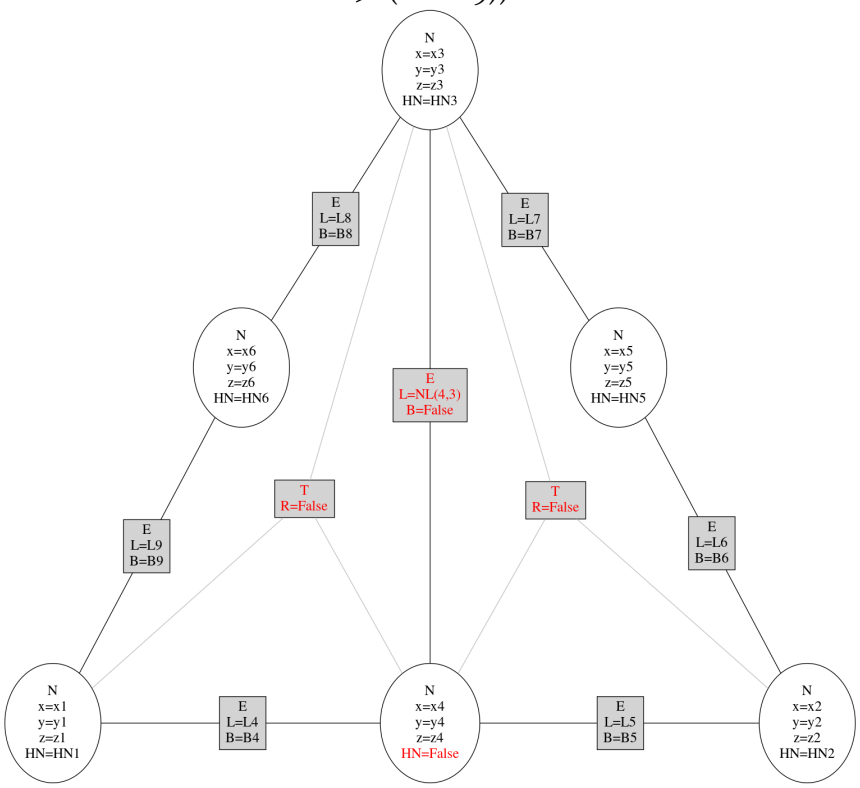


Figure 8: Production **(P6)** removing the hanging node from the longest edge of the element with three hanging nodes.

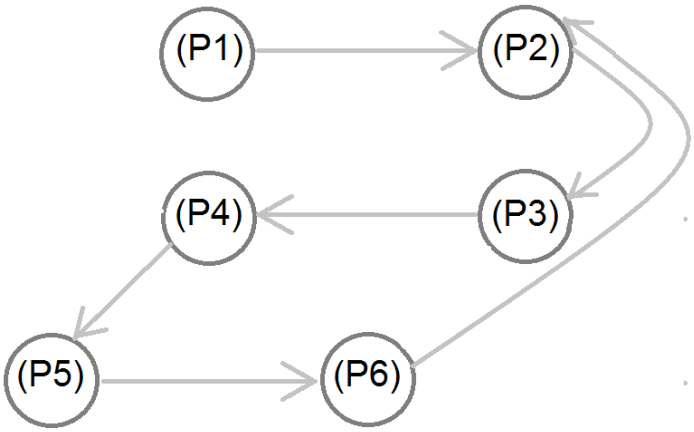


Figure 9: Control diagram executing graph grammar productions.

```
    ConnectivityManager connManager{graph};
    ProductionState pState(connManager, node, config.version2D,
                           [&map](double x, double y) ->double {
                             return map->get_height(x, y);
                           });
    for (Production *production : productions) {
      if (production->execute(pState, ctx)) {
        afterStep(i, graph);
        return;
      }
    }
  });
}
```

The next example presents the details of execution of the second production **(P2)**.

It checks first if a triangle has just one broken edge, and if the broken edge is the longest edge. If this is the case, then it breaks the triangular element.

```
bool execute(ProductionState &pState,
             galois::UserContext<GNode>&ctx) override {
  if (countBrokenEdges(pState.getEdgesIterators()) != 1) {
    return false;
  }

  int brokenEdge = pState.getAnyBrokenEdge();
  if (!checkIfBrokenEdgeIsTheLongest(brokenEdge,
```

```
                        pState.getEdgesIterators(),
                        pState.getVertices(),
                        pState.getVerticesData())) {
      return false;
    }
    breakElementWithHangingNode(brokenEdge, pState, ctx);
    return true;
}


void breakElementWithHangingNode(int edgeToBreak,
                        ProductionState &pState,
                        galois::UserContext<GNode>&ctx) const {
    GNode &hangingNode = getHangingNode(edgeToBreak, pState);
    breakElementUsingNode(edgeToBreak, hangingNode, pState, ctx);
    hangingNode->getData().setHanging(false);
}
```

This example in turn presents the detail of breaking the triangular element within production **(P2)**.

```
void breakElementUsingNode(int edgeToBreak,
                        GNode const &hangingNode,
                        const ProductionState &pState,
                        galois::UserContext<GNode>&ctx) const {
    const std::pair<int, int>&brokenEdgeVertices =
                        getEdgeVertices(edgeToBreak);
    Graph &graph = connManager.getGraph();
    int neutralVertex = getNeutralVertex(edgeToBreak);
    NodeData hNodeData = hangingNode->getData();
    double length = hNodeData.getCoords().dist(
                        pState.getVerticesData([neutralVertex]
                        .getCoords(),
                        pState.isVersion2D());
    addEdge(graph, hangingNode, pState.getVertices()[neutralVertex],
              false, length, (hNodeData.getCoords() +
                    pState.getVerticesData([neutralVertex].getCoords()) / 2);

    connManager.createInterior(hangingNode,
                        pState.getVertices()[neutralVertex],
                        pState.getVertices()[brokenEdgeVertices.first], ctx);
    connManager.createInterior(hangingNode,
```

```
                    pState.getVertices()[neutralVertex],
                    pState.getVertices()[brokenEdgeVertices.second], ctx);

  graph.removeNode(pState.getInterior());
}
```

Finally, we present the source code of some utility routines, finding the neighbors of a node `getNeighbours(GNode node)` , iterating through edges of the element `getTriangleEdges`, finding the edge between two nodes of element `getEdge`, finding the node between two vertices of broken edge `findNodeBetween`, as well as removing the edge `removeEdge`.

```
vector<GNode>getNeighbours(GNode node) const {
  vector<GNode>vertices;
  for (Graph::edge_iterator ii = graph.edge_begin(node),
    ee = graph.edge_end(node); ii != ee; ++ii) {
      vertices.push_back(graph.getEdgeDst(ii));
  }
  return vertices;
}
vector<galois::optional<EdgeIterator»getTriangleEdges(
                  vector<GNode>vertices) {
  vector<optional<EdgeIterator»edges;
  for (int i = 0; i <3; i++) {
    edges.emplace_back(getEdge(vertices[i],
                vertices[(i + 1) % 3]));
  }
  return edges;
}
galois::optional<EdgeIterator>getEdge(const GNode &v1,
                  const GNode &v2) const {
  EdgeIterator edge = graph.findEdge(v1, v2);
  return maybeEdge = convertToOptionalEdge(edge);
}


galois::optional<GNode>findNodeBetween(const GNode &node1,
                        const GNode &node2) const {
  Coordinates expectedLocation = (node1->getData().getCoords() +
                        node2->getData().getCoords()) / 2.;
  std::vector<GNode>neighbours1 = getNeighbours(node1);
```

```
    std::vector<GNode>neighbours2 = getNeighbours(node2);
    for (GNode &iNode : neighbours1) {
      auto iNodeData = graph.getData(iNode);
      for (GNode &jNode : neighbours2) {
        if (iNode == jNode &&
          iNodeData.getCoords().isXYequal(expectedLocation)) {
          return optional<GNode>(iNode);
        }
      }
    }
    return optional<GNode>();
}
void removeEdge(const GNode &node1, const GNode &node2) const {
    const EdgeIterator &edge1 = graph.findEdge(node1, node2);
    if (edge1.base() != edge1.end()) {
      graph.removeEdge(node1, edge1);
      return;
    }
    const EdgeIterator &edge2 = graph.findEdge(node2, node1);
    if (edge2.base() != edge2.end()) {
      graph.removeEdge(node2, edge2);
      return;
    }
    std::cerr «"Problem in removing an edge." «std::endl;
}
```

2.4 ADVECTION-DIFFUSION-REACTION SOLVER

2.4.1 *Tetrahedral finite elements*

On top of the topographic mesh generation, we generalize the two-dimensional triangular terrain mesh into three-dimensional tetrahedral mesh, by generating natural layers of tetrahedral elements (see Figure 10).

We construct M prismatic elements, and we divide each prismatic element into three tetrahedral elements (see Fig. 10). Next, we take a references tetrahedral element span over $(0, 0, 0) -$

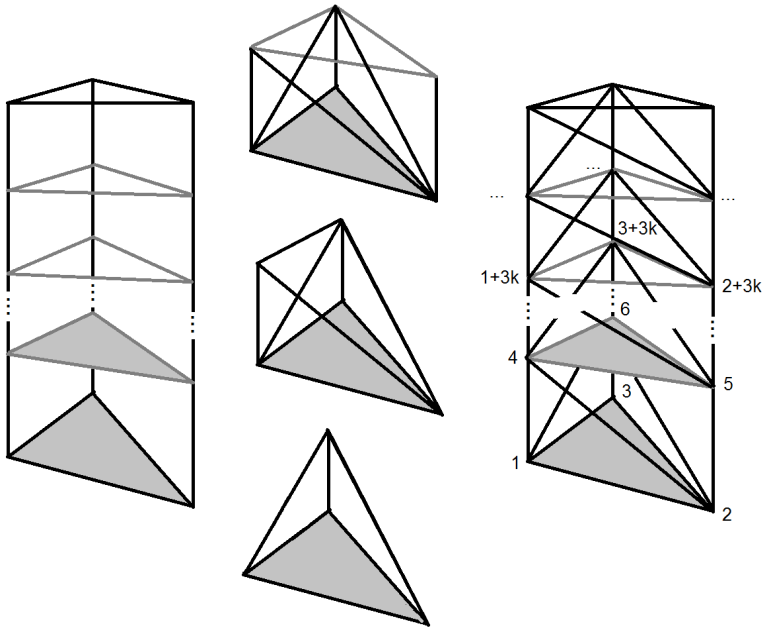


Figure 10: Generation of three dimensional mesh based on terrain 2D mesh. Generation of layers of prisms followed by the generation of tetrahedrals.

$(1,0,0)$, $(0,0,0)-(0,1,0)$, $(0,0,0)-(0,0,1)$. We introduce the four basis functions

$$\hat{\psi}_1(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3) = 1-\xi_1-\xi_2-\xi_3 \quad (1)$$
$$\hat{\psi}_2(\xi_1,\xi_2,\xi_3) = \lambda_2(\xi_1,\xi_2,\xi_3) = \xi_1 \quad (2)$$
$$\hat{\psi}_3(\xi_1,\xi_2,\xi_3) = \lambda_3(\xi_1,\xi_2,\xi_3) = \xi_2 \quad (3)$$
$$\hat{\psi}_4(\xi_1,\xi_2,\xi_3) = \lambda_4(\xi_1,\xi_2,\xi_3) = \xi_3 \quad (4)$$

related to tetrahedral element vertices, six basis functions related to finite element edges

$$\hat{\psi}_5(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3) \quad (5)$$
$$\hat{\psi}_6(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3) \quad (6)$$
$$\hat{\psi}_7(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3) \quad (7)$$
$$\hat{\psi}_8(\xi_1,\xi_2,\xi_3) = \lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3) \quad (8)$$
$$\hat{\psi}_9(\xi_1,\xi_2,\xi_3) = \lambda_2(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3) \quad (9)$$
$$\hat{\psi}_{10}(\xi_1,\xi_2,\xi_3) = \lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3), \quad (10)$$

four basis function related to finite element faces

$$\hat{\psi}_{11}(\xi_1, \xi_2, \xi_3) = \lambda_1(\xi_1, \xi_2, \xi_3)\lambda_2(\xi_1, \xi_2, \xi_3)\lambda_3(\xi_1, \xi_2, \xi_3) \tag{11}$$

$$\hat{\psi}_{12}(\xi_1, \xi_2, \xi_3) = \lambda_1(\xi_1, \xi_2, \xi_3)\lambda_2(\xi_1, \xi_2, \xi_3)\lambda_4(\xi_1, \xi_2, \xi_3) \tag{12}$$

$$\hat{\psi}_{13}(\xi_1, \xi_2, \xi_3) = \lambda_1(\xi_1, \xi_2, \xi_3)\lambda_3(\xi_1, \xi_2, \xi_3)\lambda_4(\xi_1, \xi_2, \xi_3) \tag{13}$$

$$\hat{\psi}_{14}(\xi_1, \xi_2, \xi_3) = \lambda_2(\xi_1, \xi_2, \xi_3)\lambda_3(\xi_1, \xi_2, \xi_3)\lambda_4(\xi_1, \xi_2, \xi_3), \tag{14}$$

and one basis function related to element interior

$$\hat{\psi}_{15}(\xi_1, \xi_2, \xi_3) = \lambda_1(\xi_1, \xi_2, \xi_3)\lambda_2(\xi_1, \xi_2, \xi_3) \lambda_3(\xi_1, \xi_2, \xi_3)\lambda_4(\xi_1, \xi_2, \xi_3). \tag{15}$$

The basis functions over an arbitrary element are obtained by using the transformation of the reference element to an arbitrary element.

For an exemplary discrete weak formulation

$$b(u, v) = l(v) \quad \forall v \in V \tag{16}$$

where

$$b(u, v) = \int u(x, y, z)v(x, y, z)dxdydz \tag{17}$$

$$l(v) = \int f(x, y, z)v(x, y, z)dxdydz \tag{18}$$

where $f(x, y, z)$ is given. Let K denotes tetrahedral element. Element matrices are defined as

$$b(\psi_i, \psi_j) = \int_K \psi_i(x, y, z)\psi_j(x, y, z)dxdydzi = 1, ..., 14; j = 1, ..., 14 \tag{19}$$

$$l(v_h) = (f, v_h)_\Omega \tag{20}$$

Assuming (for simplicity) $f = 1$ we get:

$$\begin{bmatrix} \int_K \psi_1(x,y,z)\psi_1(x,y,z)dxdydz & \cdots & \int_K \psi_1(x,y,z)\psi_{15}dxdydz \\ \vdots & \vdots & \vdots \\ \int_K \psi_1 5(x,y,z)\psi_1(x,y,z)dxdydz & \cdots & \int_K \psi_1 5(x,y,z)\psi_{15}dxdydz \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_{15} \end{bmatrix} = \tag{21}$$

$$\begin{bmatrix} \int_K 1 * \psi_1(x,y,z)dxdydz \\ \vdots \\ \int_K 1 * \psi_1 5(x,y,z)dxdydz \end{bmatrix} \tag{22}$$

2.4.2 *Strong form of the advection-diffusion-reaction equations*

We employ the advection-diffusion-reaction equations

$$\frac{\partial u}{\partial t} + \beta \cdot \nabla u - \nabla \cdot (\epsilon \nabla u) + cu = f \tag{23}$$

Here $u(x,y,z,t)$ is the pollutant concentration field, $\beta(x,y,z,t) = (\beta_x(x,y,z,t), \beta_y(x,y,z,t), \beta_z(x,y,z,t))$ is a given wind velocity vector field, ϵ is the diffusion coefficient, and cu is the reaction term. The advection term defines the movement of the polluted air masses forced by the wind, and the diffusion term defines the natural movement of the polluted particles in the air.

We discretize in time by introducing the time mesh $0 = t_0 < t_1 < t_2 < \cdot < t_N = T$. We employ the finite difference with respect to time with the Crank-Nicolson time integration scheme.

$$\frac{u^{t+1} - u^t}{dt} + \beta \cdot \nabla \frac{u^{t+1} + u^t}{2} - \nabla \cdot \left(\epsilon \nabla \frac{u^{t+1} + u^t}{2} \right) + c \frac{u^{t+1} + u^t}{2} = f^t \tag{24}$$

2.4.3 *Weak form of the advection-diffusion-reaction equations*

We introduce the variational formulation: We seek $u \in V = H^1(\Omega)$ such that

$$\frac{u^{t+1} - u^t}{\Delta t} + \frac{b(u^t, v) + b(u^{t+1}, v)}{2} = l(v) \quad \forall v \in V \tag{25}$$

where

$$b(u,v) = (\beta \cdot \nabla u, v)_\Omega - (\epsilon \nabla u, \nabla v)_\Omega + (\epsilon n \cdot \nabla u, v)_\Gamma + (cu, v)_\Omega \tag{26}$$

$$l(v) = (f, v)_\Omega \tag{27}$$

where $(u,v)_\Omega = \int_\Omega uvdxdydz$, and $(u,v)_\Gamma = \int_\Gamma uvds$ denotes the L^2 scalar product on Ω, $\Gamma = \partial\Omega$, and $n = (n_x, n_y, n_z)$ is the versor normal to Γ.

2.4.4 *Galerkin method*

We employ the Galerkin method with finite element discretization. We seek for $u_h \in V_h \subset V$

$$\left(\frac{u_h^{t+1} - u_h^t}{\Delta t}, v_h\right) + \frac{b(u_h^t, v_h) + b(u_h^{t+1}, v_h)}{2} = l(v_h) \quad \forall v_h \in V_h \subset V \tag{28}$$

where V_h is span by the tetrahedral finite elements and base functions obtained from glueing together the tetrahedral element base functions.

We employ the Streamline-upwind Petrov-Galerkin (SUPG) method proposed originally by Prof. Tom Hughes [37]. In this method, we modify the weak form to stabilize the formulation

$$\begin{aligned} b(u_h^{t+1}, v_h) + \sum_K (R(u_h^{t+1}), \tau\beta \cdot \nabla v_h)_K = \\ l(v_h) + \sum_K (f, \tau\beta \cdot \nabla v_h)_K \quad \forall v \in V \end{aligned} \tag{29}$$

where $R(u_h^{t+1}) = \beta \cdot \nabla u_h^{t+1} + \epsilon \Delta u_h^{t+1}$, and $\tau^{-1} = \beta \cdot \left(\frac{1}{h_K^x}, \frac{1}{h_K^y}, \frac{1}{h_K^z}\right) + 3p^2\epsilon \frac{1}{{h_K^x}^2 + {h_K^y}^2 + {h_K^z}^2}$, where ϵ stands for the diffusion term, and $\beta = (\beta_x, \beta_y, \beta_z)$ for the advection term, and h_K^x, h_K^y and h_K^z are three dimensions of an element K. Thus, we have

$$b_{SUPG}(u_h^{t+1}, v_h) = l_{SUPG}(v_h) \quad \forall v_h \in V_h \tag{30}$$

$$
\begin{aligned}
&b_{SUPG}(u_h^{t+1}, v_h) = \beta_x \left(\frac{\partial u_h^{t+1}}{\partial x}, v_h \right)_\Omega + \\
&\beta_y \left(\frac{\partial u_h^{t+1}}{\partial y}, v_h \right)_\Omega + \beta_z \left(\frac{\partial u_h^{t+1}}{\partial z}, v_h \right)_\Omega + \\
&\epsilon \left(\frac{\partial u_h^{t+1}}{\partial x}, \frac{\partial v_h}{\partial x} \right)_\Omega + \epsilon \left(\frac{\partial u_h^{t+1}}{\partial y}, \frac{\partial v_h}{\partial y} \right)_\Omega + \\
&\epsilon \left(\frac{\partial u_h^{t+1}}{\partial z}, \frac{\partial v_h}{\partial z} \right)_\Omega + (c u_h, v_h)_\Omega - \left(\epsilon \frac{\partial u_h^{t+1}}{\partial x} n_x, v_h \right)_\Gamma - \\
&\left(\epsilon \frac{\partial u_h^{t+1}}{\partial y} n_y, v_h \right)_\Gamma - \left(\epsilon \frac{\partial u_h^{t+1}}{\partial z} n_z, v_h \right)_\Gamma + \\
&\left(\beta_x \frac{\partial u_h^{t+1}}{\partial x} + \beta_y \frac{\partial u_h^{t+1}}{\partial y} + \beta_z \frac{\partial u_h^{t+1}}{\partial z} + \epsilon \Delta u_h^{t+1}, \right. \\
&\left. + \left(\frac{1}{h_x} + 3\epsilon \frac{p^2}{{h_K^x}^2 + {h_K^y}^2} \right)^{-1} \beta_x \frac{\partial v_h}{\partial x} + \beta_y \frac{\partial v_h}{\partial y} + \beta_z \frac{\partial v_h}{\partial z} \right)_\Omega
\end{aligned}
$$

$$
\begin{aligned}
l_{SUPG}(v_h) = &(f, v_h)_\Omega \\
&+ \left(f, \left(\frac{1}{h_x} + 3\epsilon \frac{p^2}{{h_K^x}^2 + {h_K^y}^2} \right)^{-1} \right. \\
&\left. \left(\beta_x \frac{\partial v_h}{\partial x} + \beta_y \frac{\partial v_h}{\partial y} + \beta_z \frac{\partial v_h}{\partial z} \right) \right)_\Omega .
\end{aligned}
$$

We incorporate the implicite Crank-Nicolson method into the finite element setup

$$
\begin{aligned}
&\left(\frac{u^{t+1} - u^t}{\Delta t}, w_h \right)_\Omega + b_{SUPG} \left(\frac{u_h^t + u_h^{t+1}}{2}, v_h \right) \\
&= l_{SUPG}(v_h) \quad \forall v_h \in V_h
\end{aligned}
$$

$$
\begin{aligned}
&\left(u^{t+1}, w_h \right)_\Omega + \frac{\Delta t}{2} b_{SUPG} \left(u_h^{t+1}, v_h \right) \\
&= \left(u^t, w_h \right)_\Omega + \frac{\Delta t}{2} b_{SUPG} \left(u_h^t, v_h \right) + l_{SUPG}(v_h) \forall v_h \in V_h
\end{aligned}
$$

The element matrices and right-hand side vectors are discretized to obtain the local systems over each element, with matrices and right-hand-sides

$$\begin{bmatrix} (\psi_1,\psi_1) & \cdots & (\psi_1,\psi_{15}) \\ \vdots & \vdots & \vdots \\ (\psi_{15},\psi_1) & \cdots & (\psi_{15},\psi_{15}) \end{bmatrix} \begin{bmatrix} u_1^t \\ \vdots \\ u_{15}^t \end{bmatrix} + \frac{\Delta t}{2} * \begin{bmatrix} b^K_{SUPG}(\psi_1,\psi_1) & \cdots & b^K_{SUPG}(\psi_1,\psi_{15}) \\ \vdots & \vdots & \vdots \\ b^K_{SUPG}(\psi_{15},\psi_1) & \cdots & b^K_{SUPG}(\psi_{15},\psi_{15}) \end{bmatrix} \begin{bmatrix} u_1^t \\ \vdots \\ u_{15}^t \end{bmatrix} = \begin{bmatrix} l^K_{SUPG}(\psi_1) \\ \vdots \\ l^K_{SUPG}(\psi_{15}) \end{bmatrix} \tag{31}$$

The resulting local systems are submitted to the GMRES iterative solver.

We propose the following space refinement - time progression algorithm. We start with an initial grid approximating roughly the topography of the terrain. We solve the first time step of the advection-diffusion-reaction problem with initial conditions. We then perform one iteration of the longest edge refinement algorithm. We then solve the second time step of the advection-diffusion-reaction problem using the solution obtained in the previous step on a coarser grid. We continue the spatial iterations of the longest edge refinement algorithm while iterating with the time step with simulations of the advection-diffusion reactions. The general idea of the algorithm can be summarized in the pseudocode presented in Algorithm 3. Note that we assume a certain accuracy of the *epsilon* terrain approximation, and once this accuracy is achieved, we do not perform further corrections there.

Algorithm 1 Space refinement - time progression algorithm

Require: ϵ, *mesh*, *initial_configuration*

```
previous_u:=initial_configuration
Refine mesh with accuracy ε
for time_step=1,...,MAX_TIME_STEP do
    Build element matrices and right-hand-side vectors
    Project previous_u into the mesh
    Aggregate the local systems into global matrix
    Solve the global system using GMRES iterative solver:
    call GMRES solver to get current_solution
    previous_u:=current_solution
end for
```

2.5 ATMOSPHERIC SIMULATIONS BASED ON ALTERNATING DIRECTION SOLVER WITH FINITE DIFFERENCE METHOD

This section presents an alternative approach to modeling the propagation of pollutants in the atmosphere.

We present a solver that uses a directional division technique and stabilized time integration schemes to solve the three-dimensional non-stationary Navier-Stokes-Boussinesq equations. The model can be used to model atmospheric phenomena. It is possible to discretize the equations with a time integration scheme such that the Kronecker product structure of the matrix is obtained. This allows for an efficient direction splitting algorithm with a finite difference solver [29], since the matrix is the Kronecker product of three triangular matrices resulting from the discretization along the x, y, and z axes. The directional division solver is not an iterative solver; it is equivalent to the Gauss elimination algorithm.

We show how to extend the alternating directions solver to irregular geometries, including terrain data while keeping the computational cost of the solver linear. We follow an idea originally used in [40] for sequential particle flow computations. We show how to modify the algorithm to work on complex irregular terrain structures and still maintain a linear computational cost.

Thus, as we showed in [29], if well parallelized, the parallel factorization cost is close to $\mathcal{O}(N/c)$ in each time step, where N is the number of unknowns and c is the number of cores. We analyze the parallel scalability of the code to 1024 multicore processors on the PROMETHEUS Linux cluster [6] from the CYFRONET supercomputing center. Each subdomain is processed using a finite-difference grid of $50 \times 50 \times 100$. Our code is coupled with a grid generator [67] that reads the NASA [17] database and provides a map of the Earth's terrain. The goal is to compare an algorithm for solving problems with alternating directions with an adaptive approach based on graph grammar.

We focus on the comparison of the algorithm to the graph-grammar-based one. We leave the detailed formulation of the model of various atmospheric phenomena in the alternating direction method for future work.

2.5.1 *Alternating directions solver for atmospheric simulations*

We now focus on modeling atmospheric phenomena using the Navier-Stokes-Boussinesq equations, using a solver with alternating directions. These formulations are based on the work of Peter Minev and Jean Luc-Guermond [5, 30].

The equations in the strong form are

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p + \mathrm{Pr}\Delta\mathbf{u} = g\mathrm{PrRa}T + f \text{ in } \Omega\times(0,T_f] \quad (32)$$

$$\nabla\cdot\mathbf{u} = 0 \text{ in } \Omega\times(0,T_f] \quad (33)$$

$$\mathbf{u} = 0 \text{ in } \partial\Omega\times(0,T_f] \quad (34)$$

$$\frac{\partial T}{\partial t} + (u\cdot\nabla)T + \Delta T = 0 \text{ in } \Omega\times(0,T_f] \quad (35)$$

$$T = 0 \text{ in } \partial\Omega\times(0,T_f] \quad (36)$$

where u is the velocity vector field, p is the pressure, $\mathrm{Pr} = 0.7$ is the Prandt number, $g = (0,0,-1)$ is the gravity force, $\mathrm{Ra} = 1000.0$ is the Rayleigh number, T is the temperature scalar field.

We discretize using a finite difference method in space and a time-stable integration scheme, resulting in the product structure of the Kronecker matrix.

For the temperature equation and the Navier-Stokes equation, we use an unconditionally time-stable second-order integration scheme with a predictor-corrector scheme for the pressure. For example, we can use the Douglass-Gunn scheme [38] by making a uniform division of the time interval $\bar{I} = [0,T]$ as

$$0 = t_0 < t_1 < \ldots < t_{N-1} < t_N = T,$$

and denoting $\tau := t_{n+1} - t_n$, $\forall n = 0,\ldots,N-1$. In the Douglas-Gunn scheme, we integrate the solution from time step t_n to t_{n+1} in three sub-steps as follows:

$$\begin{cases} (1+\frac{\tau}{2}\mathcal{L}_1)u^{n+1/3} = \tau f^{n+1/2} + (1-\frac{\tau}{2}\mathcal{L}_1 - \tau\mathcal{L}_2 - \tau\mathcal{L}_3)u^n, \\ (1+\frac{\tau}{2}\mathcal{L}_2)u^{n+2/3} = u^{n+1/3} + \frac{\tau}{2}\mathcal{L}_2 u^n, \\ (1+\frac{\tau}{2}\mathcal{L}_3)u^{n+1} = u^{n+2/3} + \frac{\tau}{2}\mathcal{L}_3 u^n. \end{cases} \quad (37)$$

For the Navier-Stokes-Boussinesq equations, $\mathcal{L}_1 = \partial_{xx}$, $\mathcal{L}_2 = \partial_{yy}$, and $\mathcal{L}_3 = \partial_{zz}$. The forcing term represents $g\mathrm{PrRa}T^{n+1/2}$ plus the advective flow and the pressure terms $\left(u^{n+1/2}\cdot\nabla u^{n+1/2}\right) + \nabla\tilde{p}^{n+1/2}$ treated explicitly as well. The pressure is computed

with the predictor/corrector scheme. The predictor step is the following

$$\tilde{p}^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n-\frac{1}{2}}, \tag{38}$$

with $p^{-\frac{1}{2}} = p_0$ and $\phi^{-\frac{1}{2}} = 0$ computes the pressure to be used in the velocity computations. The penalty steps are the following

$$\begin{cases} \psi - \partial_{xx}\psi = -\dfrac{1}{\tau}\nabla \cdot u^{n+1}, \\ \xi - \partial_{yy}\xi = \psi, \\ \phi^{n+1/2} - \partial_{zz}\phi^{n+1/2} = \xi, \end{cases} \tag{39}$$

The corrector step updates the pressure field using the velocity results and the penalty step

$$p^{n+\frac{1}{2}} = p^{n-\frac{1}{2}} + \phi^{n+\frac{1}{2}} - \chi\nabla \cdot \left(\frac{1}{2}(u^{n+1} + u^{n})\right). \tag{40}$$

.

For the temperature equation, $\mathcal{L}_1 = \partial_{xx}$, $\mathcal{L}_2 = \partial_{yy}$, and $\mathcal{L}_3 = \partial_{zz}$. The forcing term represents the advection term treated explicitly $\left(u^{n+1/2} \cdot \nabla T^{n+1/2}\right)$.

The mathematical details of the problem formulation is described in [5, 30].

Each of the equation in our system contains only derivatives in one direction. This means that the matrices after the discretization are three-diagonal and can be factorized in a linear cost

$$\begin{aligned} (1 + \alpha\partial_{xx})\, u^{n+1/3} &= RHS_x \\ (1 + \alpha\partial_{yy})\, u^{n+2/3} &= RHS_y \\ (1 + \alpha\partial_{zz})\, u^{n+1} &= RHS_z \end{aligned} \tag{41}$$

$$\begin{aligned} u_{ijk}^{n+1/3} + \alpha\frac{u_{(i-1)jk}^{n+1/3} - 2u_{ijk}^{n+1/3} + u_{(i+1)jk}^{n+1/3}}{dt} &= RHS_x \\ u_{ijk}^{n+2/3} + \alpha\frac{u_{i(j-1)k}^{n+2/3} - 2u_{ijk}^{n+2/3} + u_{i(j+1)k}^{n+1/3}}{dt} &= RHS_y \\ u_{ijk}^{n+1} + \alpha\frac{u_{ij(k-1)}^{n+1} - 2u_{ijk}^{n+1} + u_{ij(k+1)}^{n+1}}{dt} &= RHS_z, \end{aligned} \tag{42}$$

where $\alpha = \tau/2$ or $\alpha = 1$, depending which equation we discretize. This is equivallent to

$$\begin{aligned} \alpha u^{n+1/3}_{(i-1)jk} + (dt - 2\alpha) u^{n+1/3}_{ijk} + \alpha u^{n+1/3}_{(i+1)jk} &= dt * RHS_x \\ \alpha u^{n+2/3}_{i(j-1)k} + (dt - 2\alpha) u^{n+2/3}_{ijk} + \alpha u^{n+1/3}_{i(j+1)k} &= dt * RHS_y \\ \alpha u^{n+1}_{ij(k-1)} + (dt - 2\alpha) u^{n+1}_{ijk} + \alpha u^{n+1}_{ij(k+1)} &= dt * RHS_z, \end{aligned} \tag{43}$$

All these systems have a Kronecker product structure $\mathcal{M} = \mathcal{A}^x \otimes \mathcal{B}^y \otimes \mathcal{C}^z$ where the sub-matrices are aligned along the three axis of the system of coordinates. All these sub-matrices are three-diagonal, or scaled identity matrices. The three-diagonal matrices can be factorized in a linear cost using the Thomas algorithm.

2.5.2 *Algorithm for introduction of terrain into alternating direction solver with finite difference method*

We show how to extend the alternating directions solver to irregular geometries, including terrain data while keeping the computational cost of the solver linear. We follow an idea originally used in [40] for sequential particle flow computations. We show how to modify the algorithm to work on complex irregular terrain structure and still maintain a linear computational cost.

The direction splitting algorithm for the Kronecker product matrix implements three steps, which is equivalent to the Gauss elimination algorithm [23], since

$$(\mathcal{M})^{-1} = (\mathcal{A}^x \otimes \mathcal{B}^y \otimes \mathcal{C}^z)^{-1} = (\mathcal{A}^x)^{-1} \otimes (\mathcal{B}^y)^{-1} \otimes (\mathcal{C}^z)^{-1} \tag{44}$$

We solve first along x direction

$$\begin{bmatrix} A^x_{11} & A^x_{12} & \cdots & 0 \\ A^x_{21} & A^x_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A^x_{kk} \end{bmatrix} \begin{bmatrix} z_{111} & z_{121} & \cdots & z_{1lm} \\ z_{211} & z_{221} & \cdots & z_{2lm} \\ \vdots & \vdots & \ddots & \vdots \\ z_{k11} & z_{k21} & \cdots & z_{klm} \end{bmatrix} = \begin{bmatrix} y_{111} & y_{121} & \cdots & y_{1lm} \\ y_{211} & y_{221} & \cdots & y_{2lm} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k11} & y_{k21} & \cdots & y_{klm} \end{bmatrix} \tag{45}$$

We solve second along y direction

$$\begin{bmatrix} B^y_{11} & B^y_{12} & \cdots & 0 \\ B^y_{21} & B^y_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B^y_{ll} \end{bmatrix} \begin{bmatrix} y_{111} & y_{211} & \cdots & y_{k1m} \\ y_{121} & y_{221} & \cdots & y_{k2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1l1} & y_{1l1} & \cdots & y_{klm} \end{bmatrix} = \begin{bmatrix} z_{111} & z_{211} & \cdots & z_{k1m} \\ z_{121} & z_{221} & \cdots & z_{k2m} \\ \vdots & \vdots & \ddots & \vdots \\ z_{1l1} & z_{2l1} & \cdots & z_{klm} \end{bmatrix} \tag{46}$$

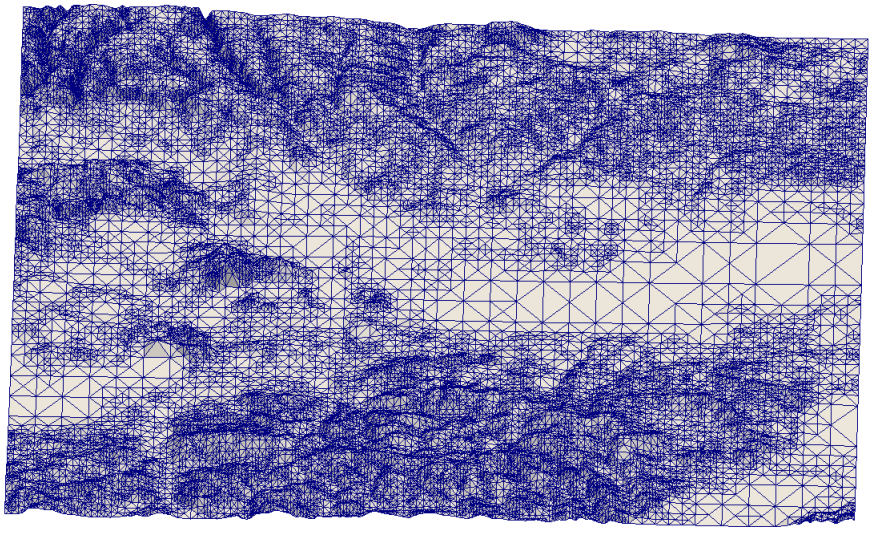


Figure 11: The computational mesh generated based on the NASA database, representing the topography of the Krakow area.

We solve third along z direction,

$$\begin{bmatrix} C^z_{1,1} & C^z_{1,2} & \cdots & 0 \\ C^z_{2,1} & C^z_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C^z_{m,m} \end{bmatrix} \begin{bmatrix} x_{111} & x_{211} & \cdots & x_{kl1} \\ x_{112} & x_{212} & \cdots & x_{kl2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{11m} & x_{21m} & \cdots & x_{klm} \end{bmatrix} = \begin{bmatrix} b_{111} & b_{211} & \cdots & b_{kl1} \\ b_{112} & b_{212} & \cdots & b_{kl2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{11m} & b_{21m} & \cdots & b_{klm} \end{bmatrix} \quad (47)$$

We combine the alternating-directions solver with a mesh generator that provides an excellent approximation of the topography of the area [67], based on the NASA [17] database. The resulting mesh generated for the Krakow area is shown in Figure 11.

We check if a given point is located inside the domain when simulating atmospheric phenomena, such as thermal inversion over a designated terrain, using an alternating direction solver and a finite difference method. Points inside the terrain (outside the atmospheric domain) are removed from the system of equations. This is done by identifying the indexes of the points along the x, y, and z axes. We then modify the systems of equations so that the corresponding three rows in the three systems of equations are reset to 0, the diagonal is set to 1, and the corresponding rows and columns of the three right-hand sides are set to 0.

For example, if we want to remove the point (r, s, t) from the system, we modify the following in the first system. We factor-

ize the column "st" in the first system separately, replacing the row in the matrix with an identity on the diagonal and using 0 on the right-hand side. The remaining columns in the first system are treated in the same way.

$$\begin{bmatrix} A^x_{11} & A^x_{12} & \cdots & 0 \\ A^x_{21} & A^x_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & A^x_{rr} = 1.0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A^x_{kk} \end{bmatrix} \begin{bmatrix} z_{1st} \\ z_{2st} \\ \vdots \\ z_{rst} \\ \vdots \\ z_{kst} \end{bmatrix} = \begin{bmatrix} y_{1st} \\ y_{2st} \\ \vdots \\ y_{rst} = 0.0 \\ \vdots \\ y_{kst} \end{bmatrix} \tag{48}$$

We factorize the column "rt" in the second system separately, setting the row in the matrix as an identity on the diagonal and using 0.0 on the right side. The other columns in the second system are treated in the same way.

$$\begin{bmatrix} B^y_{11} & B^y_{12} & \cdots & 0 \\ B^y_{21} & B^y_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & B^y_{ss} = 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & B^y_{ll} \end{bmatrix} \begin{bmatrix} y_{r1t} \\ y_{r2t} \\ \vdots \\ y_{rst} \\ \vdots \\ y_{rlt} \end{bmatrix} = \begin{bmatrix} z_{r1t} \\ z_{r2t} \\ \vdots \\ z_{rst} = 0.0 \\ \vdots \\ z_{rlt} \end{bmatrix} \tag{49}$$

In a similar way, in the third system, we decompose the column "rs" separately. The other columns in the third system are also treated in the same way.

$$\begin{bmatrix} C^z_{1,1} & C^z_{1,2} & \cdots & 0 \\ C^z_{2,1} & C^z_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & C^z_{tt} = 1.0 & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C^z_{m,m} \end{bmatrix} \begin{bmatrix} x_{rs1} \\ x_{rs2} \\ \vdots \\ x_{rst} \\ \vdots \\ x_{rsm} \end{bmatrix} = \begin{bmatrix} b_{rs1} \\ b_{rs2} \\ \vdots \\ b_{rst} = 0.0 \\ \vdots \\ b_{rsm} \end{bmatrix} \tag{50}$$

Using this trick for all points in the terrain, we can decompose the Kronecker product system into a linear computational cost in complex terrain geometry.

Part III

NUMERICAL EXPERIMENTS

3

NUMERICAL RESULTS

In this chapter we present seven numerical experiments.

1. The first one concerns the graph-grammar-based generation of the two-dimensional mesh covering the topography of Kraków.
2. The second one presents the concurrency of the classical and graph-grammar-based implementation of the longest-edge refinement algorithm.
3. The third one describes the experiment with manufactured solution, performed for the verification of the mesh generator and the solver algorithm.
4. In the next experiment, we generate topographic mesh covering the Lesser district of Poland by using the graph-grammar-based longest-edge refinement algorithm.
5. Finally, we execute large simulations of the pollution propagation in Lesser district of Poland
6. We also compare with numerical experiments concerning the alternating directions solver.

3.1 THE GRAPH-GRAMMAR BASED LONGEST-EDGE REFINEMENT ALGORITHM FOR GENERATION OF THE TOPOGRAPHY OF THE KRAKOW AREA

The goal of the first experiment is to check how the graph-grammar-based algorithm approximates the real terrain data. We selected the valley of Kraków. We start with a regular two-dimensional grid with twenty four triangles. The grid covers a topographic area of size 42954.3 (+/- 446.3) x 33354.7 (+/- 574.8) meters. This grid is not exactly rectangular because the Earth is not flat, and the input data is taken from the Earth [16] database, so the vertices of triangles are lifted to the Earth level. The algorithm performs the fourteen iterations, shown in Figures 12-24.

The algorithm transforms the initial grid of twenty four triangles into a final grid with 49,112 triangles. In the final mesh, the

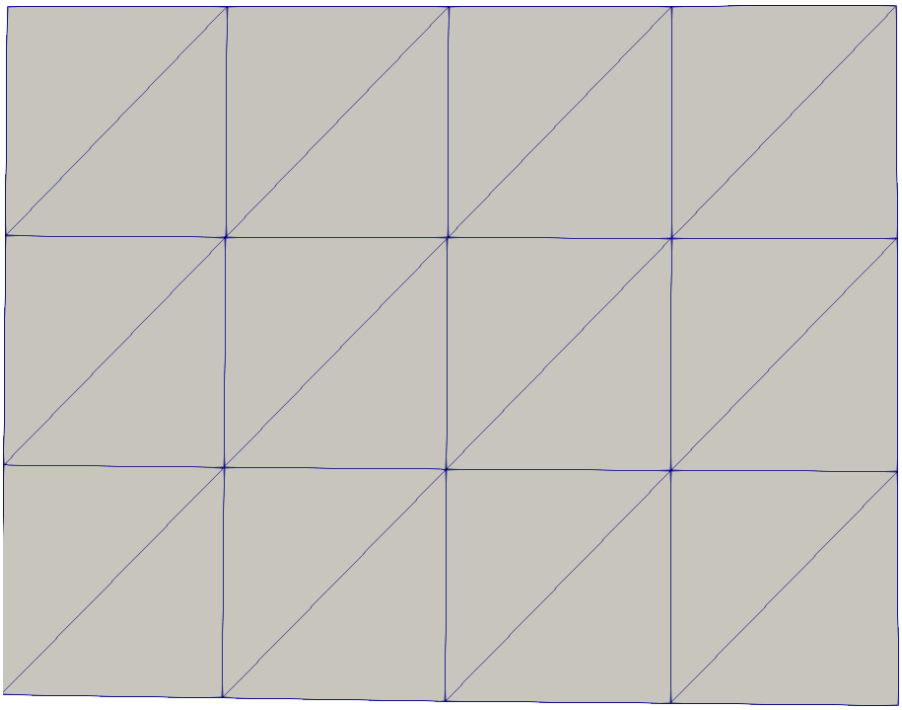


Figure 12: The uniform initial mesh with 24 elements

diameter of the smallest triangular element is 120.46m, and the diameter of the largest triangular element is 3855.98m.

3.2 COMPARISON OF THE LONGEST-EDGE REFINEMENT ALGORITHM AND GRAPH-GRAMMAR-BASED REFINEMENT ALGORITHM

In this section, we compare two algorithms, Rivara's classical longest-edge refinement algorithm and the graph-grammar-based refinement algorithm. To describe Rivara's algorithm, let's recall the definition of $LEPP(t)$, the definition of a pair of terminal triangles, and the definition of a terminal triangle. For any triangle t_0 of any conforming triangulation T, $LEPP(t_0)$ consists of an ordered list of all triangles $t_0, t_1, t_2, ..., t_{n-1}$ such that triangle t_i is a neighboring triangle t_{i-1} by the longest edge of t_{i-1}, for $i = 1, 2, ..., n$.
A pair of end triangles is two adjacent triangles (t, t^*) with a common longest edge. An end boundary triangle is a triangle whose longest edge is a boundary edge. We describe the pseudocode of the so-called Rivara Backward-Longest-Edge-Bisection algorithm in Algorithm 1.

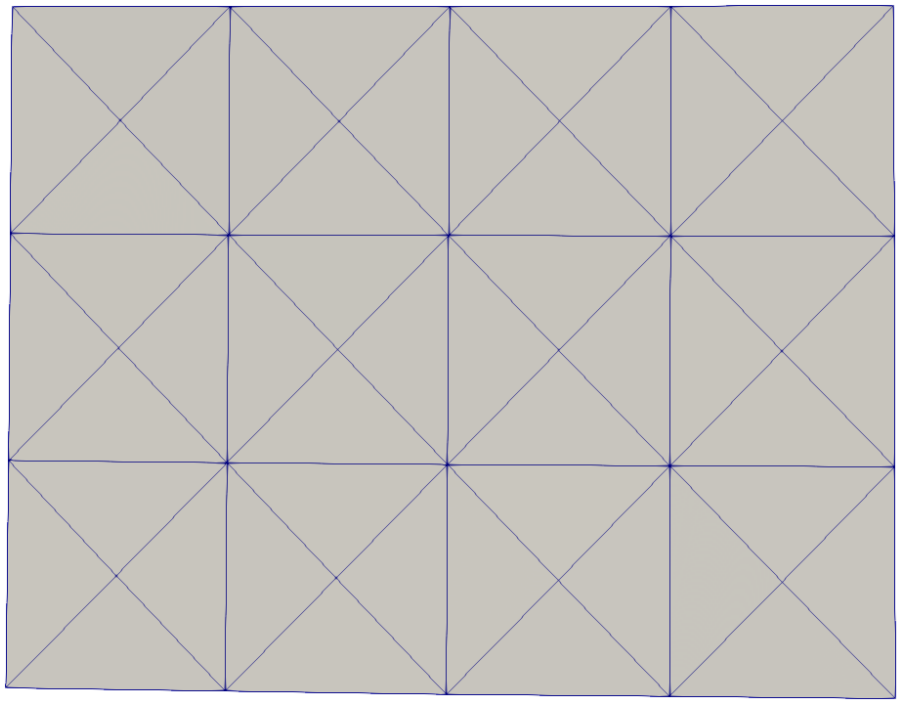


Figure 13: The first step of the graph-grammar-based refinement algorithm

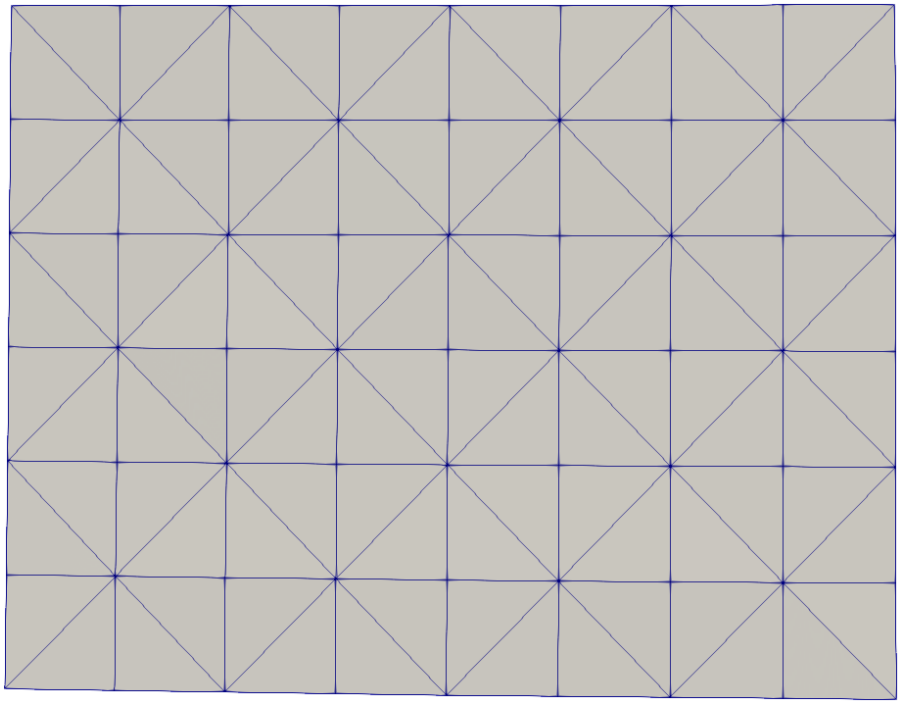


Figure 14: The second step of the graph-grammar-based refinement algorithm

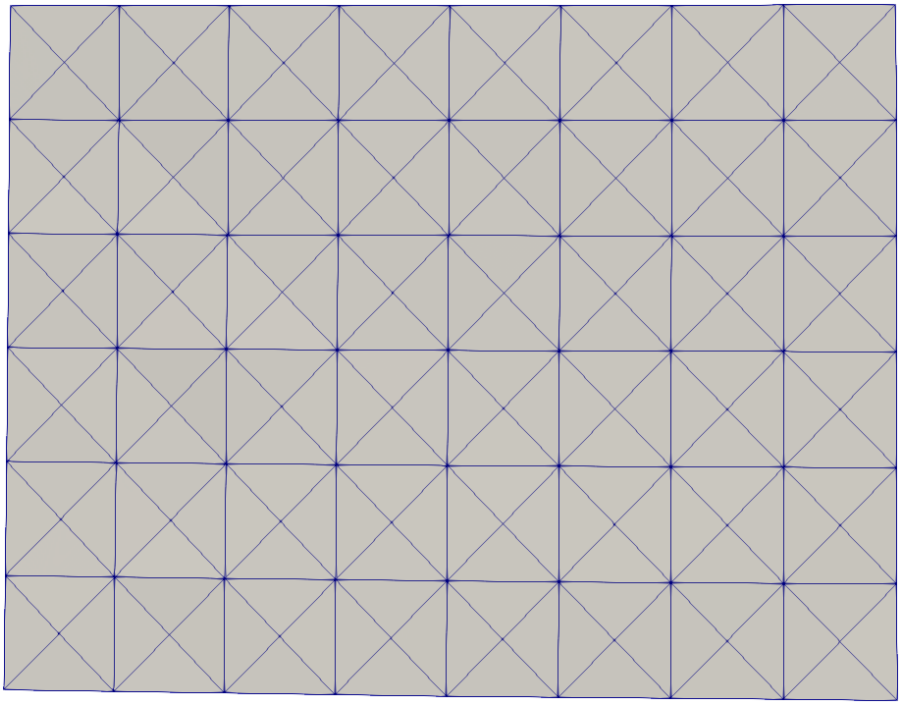


Figure 15: The third step of the graph-grammar-based refinement algorithm

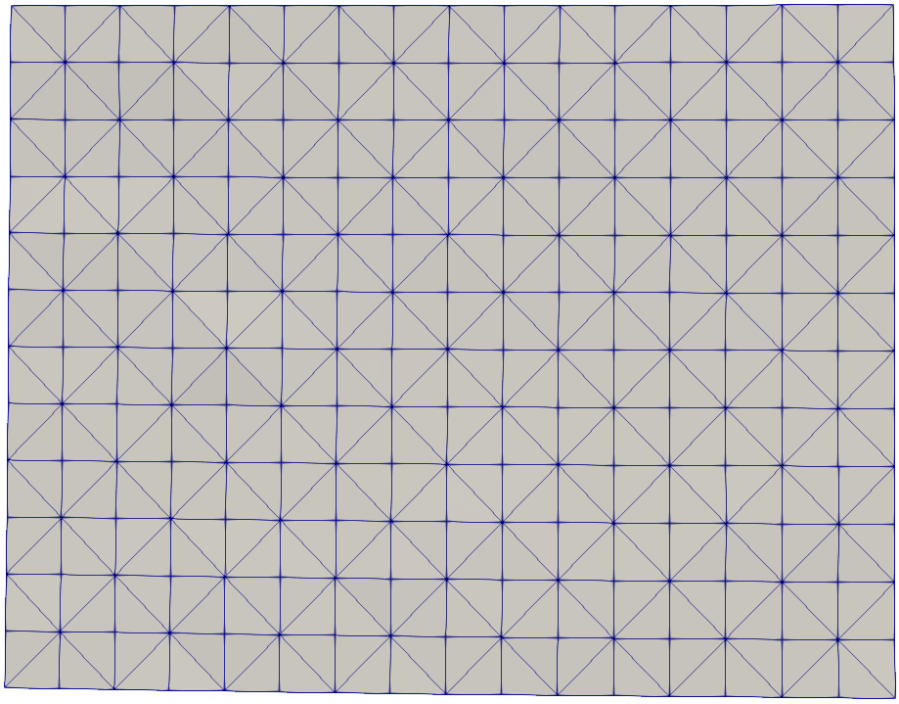


Figure 16: The fourth step of the graph-grammar-based refinement algorithm

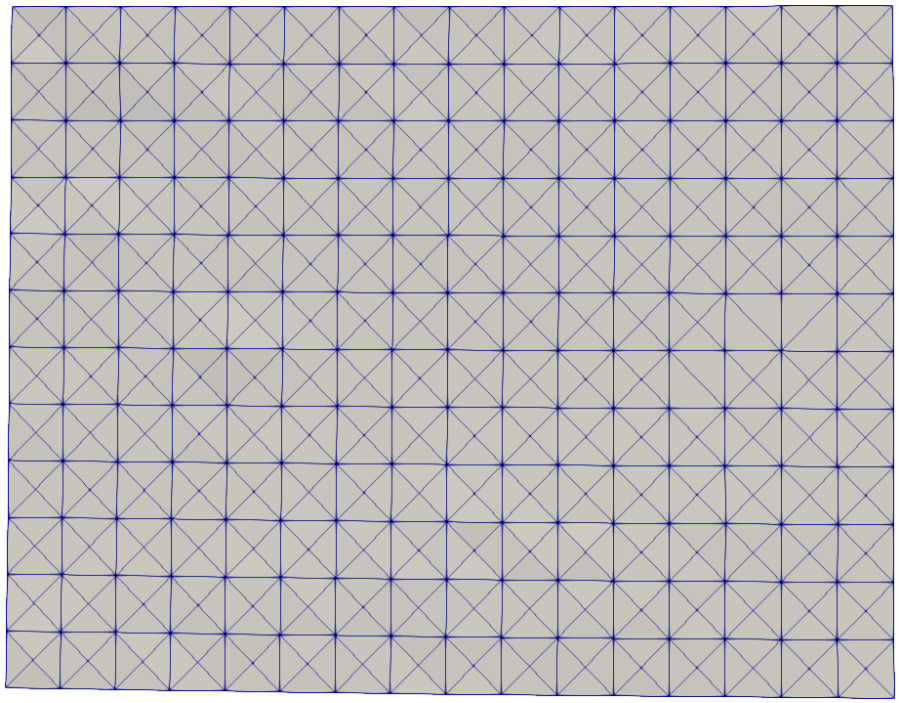


Figure 17: The fifth step of the graph-grammar-based refinement algorithm

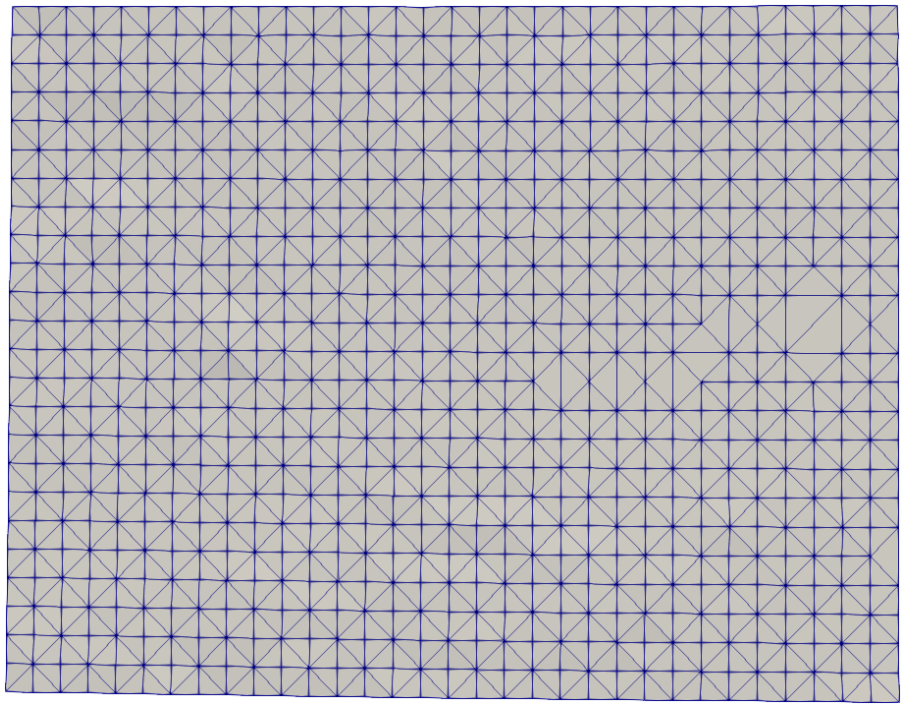


Figure 18: The sixth step of the graph-grammar-based refinement algorithm

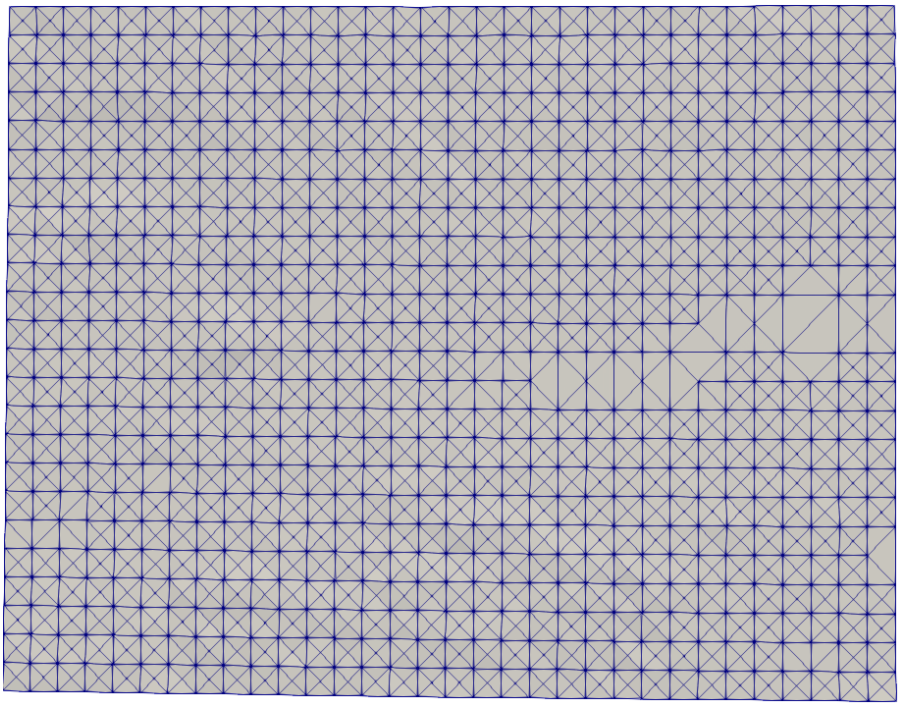


Figure 19: The seventh step of the graph-grammar-based refinement algorithm

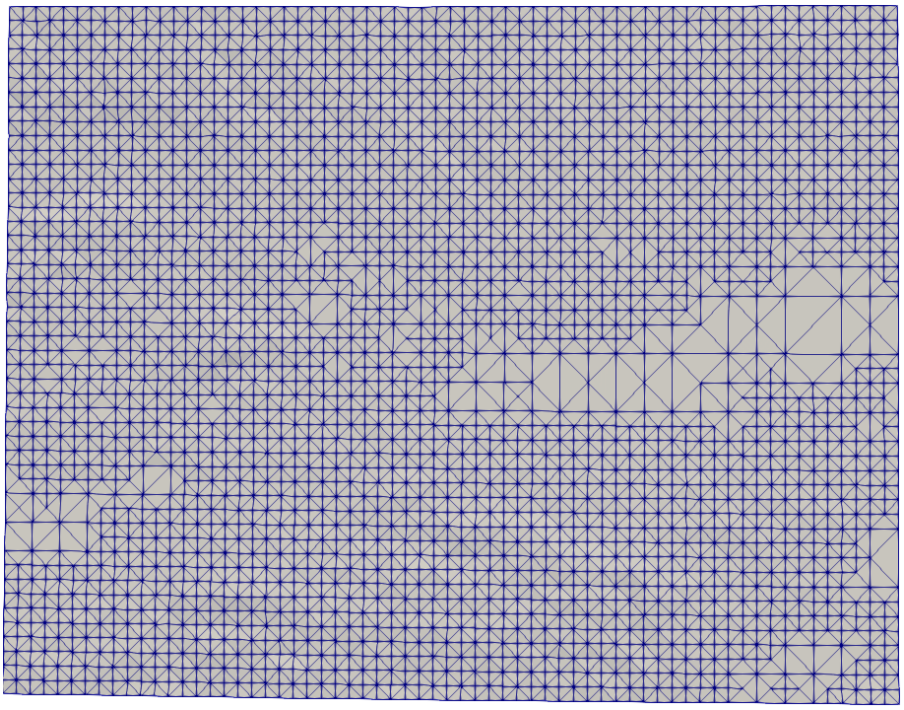


Figure 20: The eighth step of the graph-grammar-based refinement algorithm

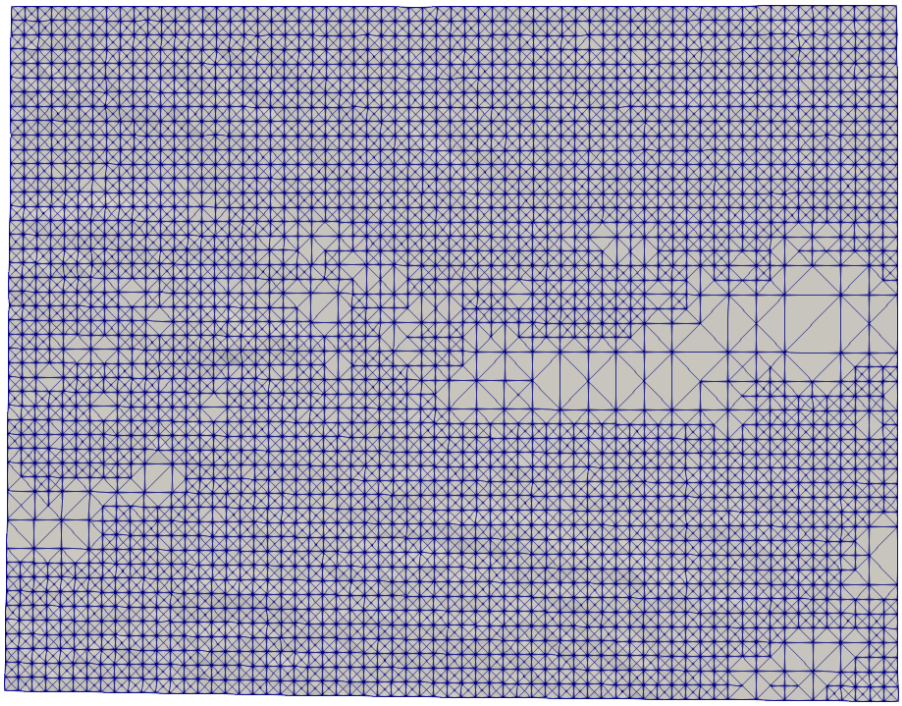


Figure 21: The ninth step of the graph-grammar-based refinement algorithm

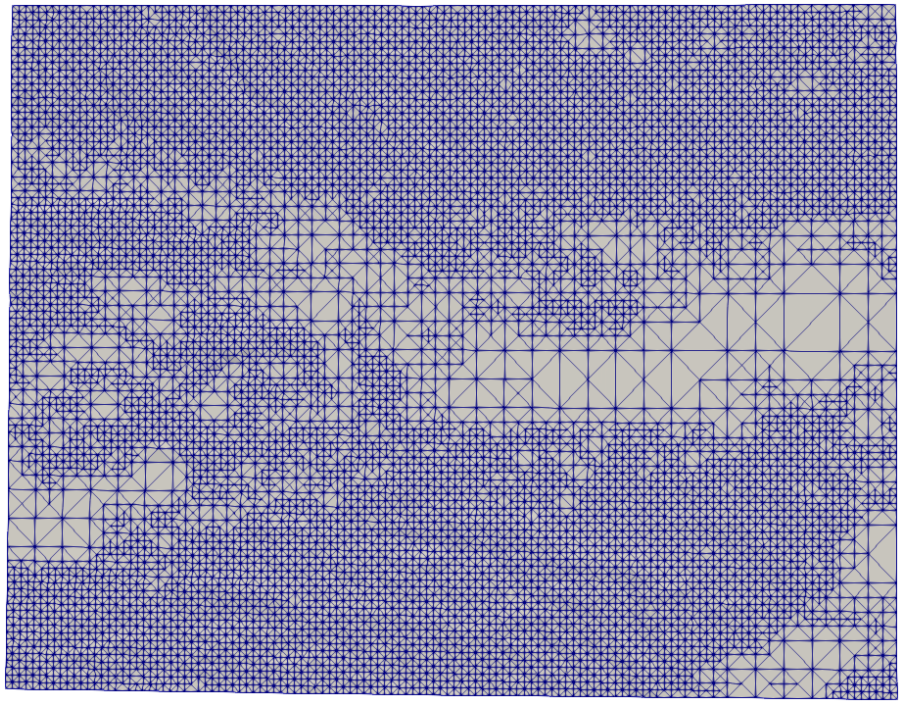


Figure 22: The ten step of the graph-grammar-based refinement algorithm

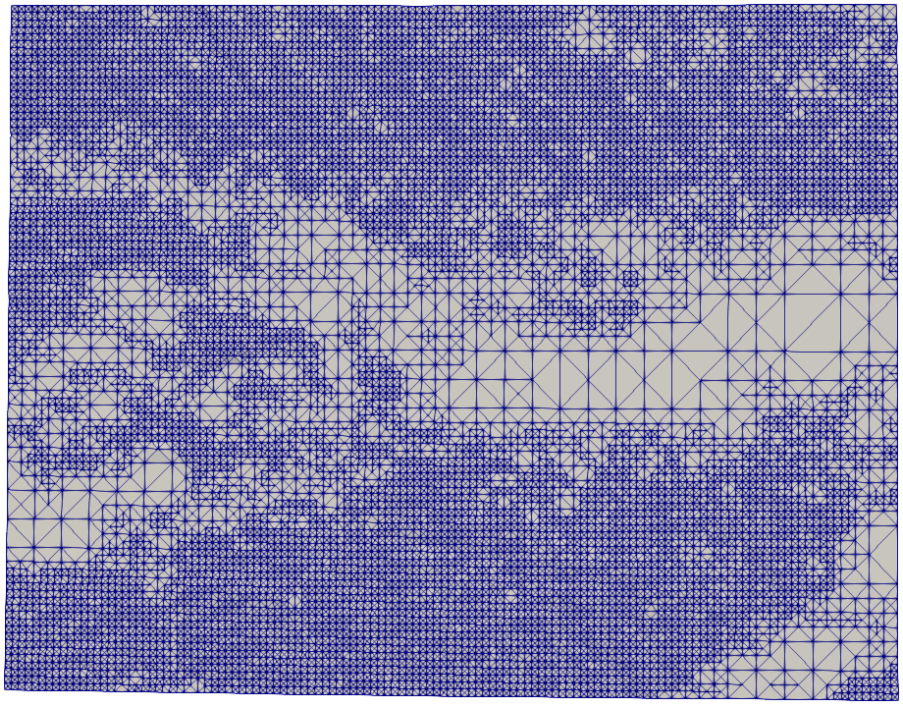


Figure 23: The eleventh step of the graph-grammar-based refinement algorithm

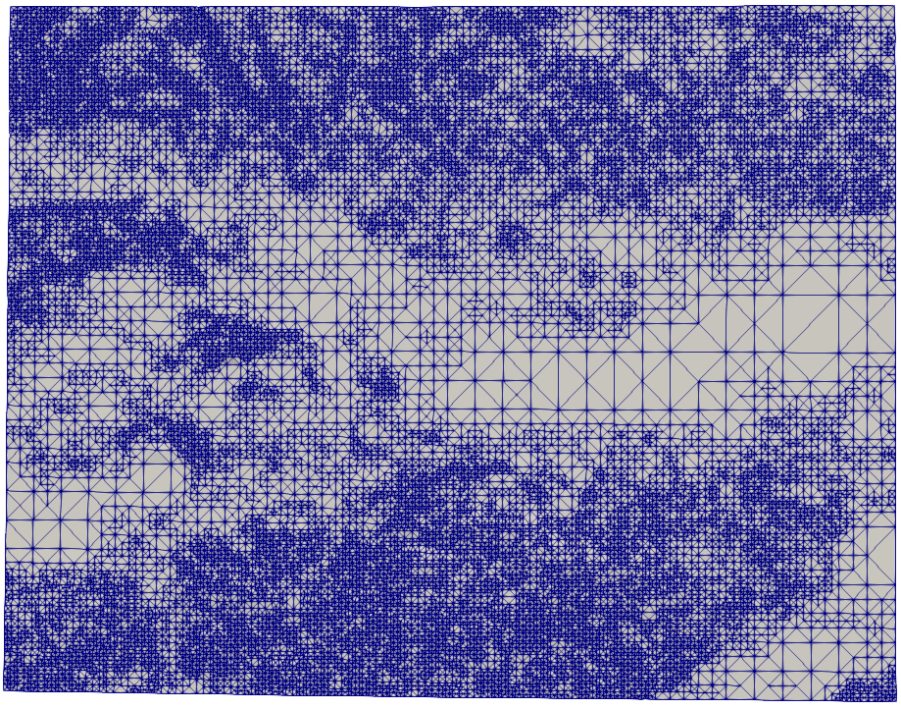


Figure 24: The twelfth step of the graph-grammar-based refinement algorithm

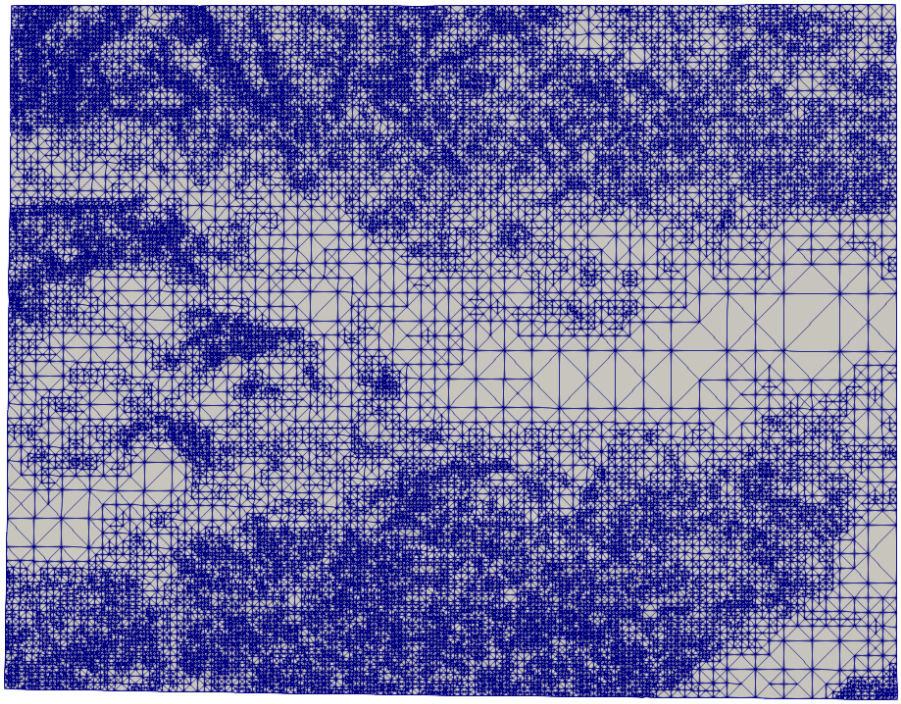


Figure 25: The thirteen step of the graph-grammar-based refinement algorithm

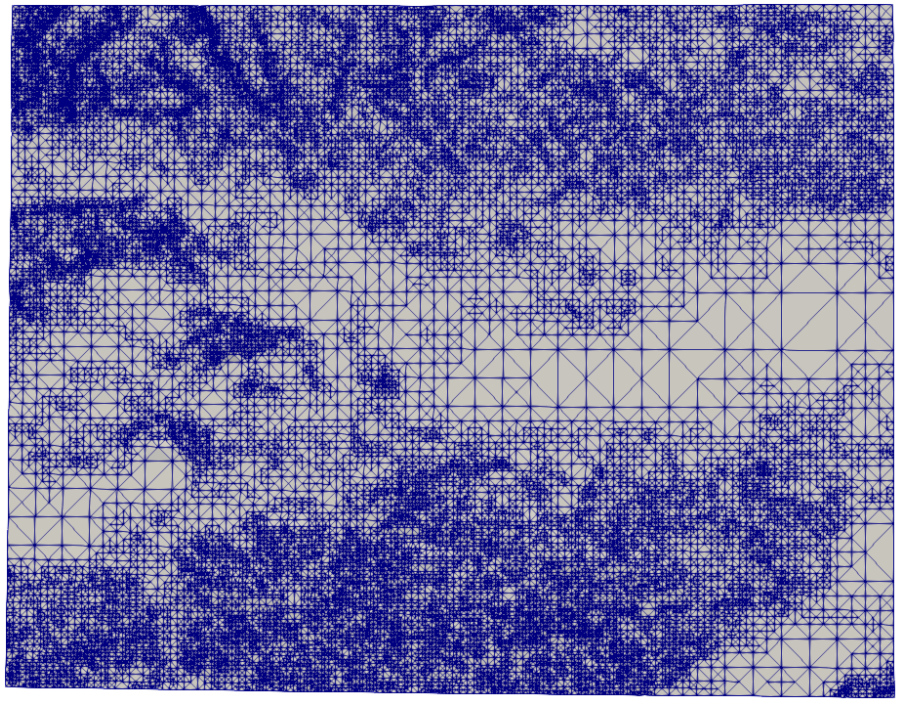


Figure 26: The fourteen step of the graph-grammar-based refinement algorithm

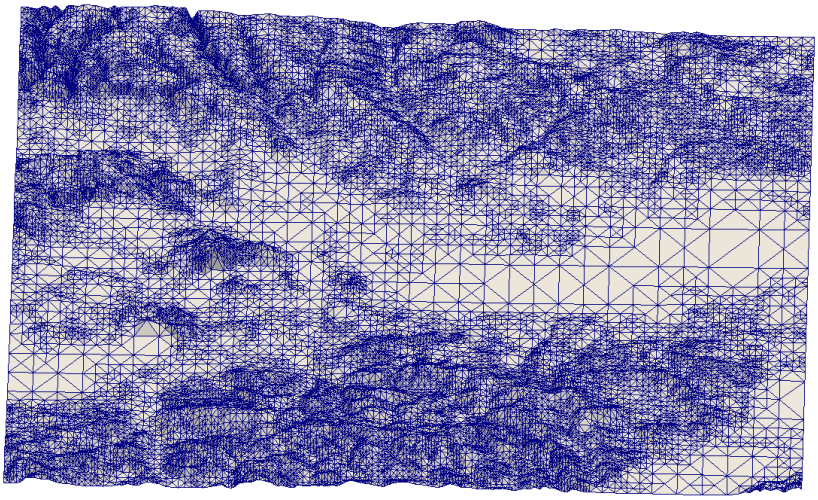


Figure 27: The three-dimensional view on the generated topographic mesh

Algorithm 2 Backward-Longest-Edge-Bisection

Require: t *triangle selected to refinement,* T *triangular elements mesh*

while Triangle t remains unbroken **do**
  Find $LEPP(t)$
  $t^* =$ the last triangle of $LEPP(t)$
  if t^* is a terminal boundary triangle **then**
    break t^*
  else
    break the last pair of terminal triangles of $LEPP(t)$
end while

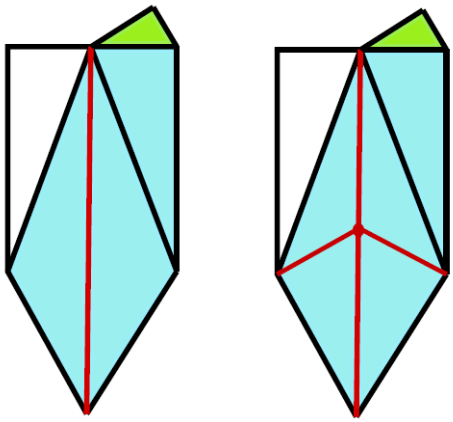


Figure 28: The first step of the Rivara algorithm.

In Figure 28, we summarize Rivara's algorithm. The green triangle is the triangle labeled as interrupted (t_0). Triangles belonging to $LEPP(t_0)$ are colored blue (triangles "affected" by the algorithm), red edges are final edges, new edges created during the refinement process. We count the number of basic operations performed by the algorithm, defined as checking a triangle (*CHECK*) (triangles colored blue) or breaking a triangle (*BREAK*) (triangles broken at the end of *LEPP*). The number of *CHECK*s and *BREAK*s is summarized in the table 1. Although the single longest edge path algorithm does not have parallelization, all these *CHECK*s are executed sequentially in each step. The algorithm for refining the longest for a single *LEPP* is sequential. Even breaking two triangles located at the end of a path is sequential, since the common edge must be locked until the first break is completed. The parallel processing time is identical. Parallelism in the classical Rivara algorithm is achieved by parallelizing multiple *LEPP*, each *LEPP* in a sequence of [74].

In Figures 34 we summarize the graph-grammar-based algorithm.

The initially broken triangle is marked in green. The marked neighbors are marked in blue. The red break line marks the broken triangles.

In the table 2, we count the number of triangles in which we attempted productions (*CHECK*) and the number of triangles modified by productions (*BREAK*s). The graph-grammar algorithm has the potential to be parallelized even when Rivary's algorithm uses a single *LEPP*. The number of sequential *CHECK*s for the graph-grammar based algorithm is 44, but when we use eight cores, this is equal to 10 steps. The number of *BREAK*s can

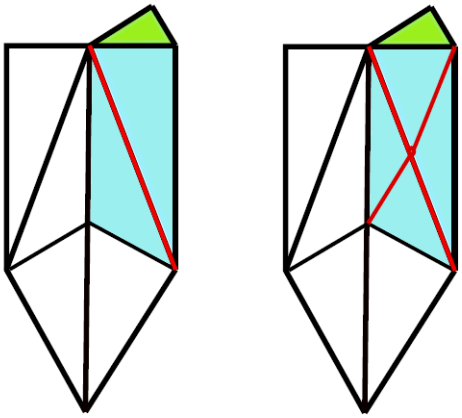


Figure 29: The second step of the Rivara algorithm.

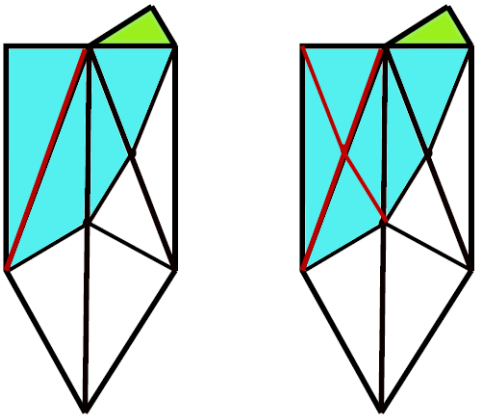


Figure 30: The third step of the Rivara algorithm.

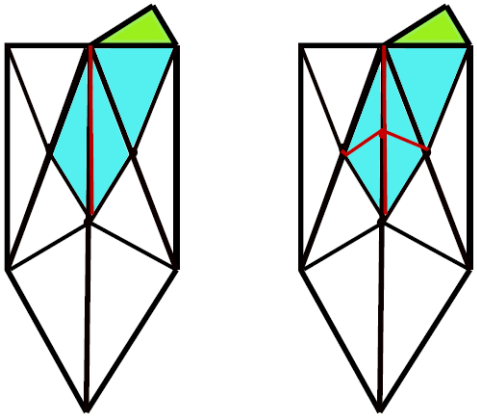


Figure 31: The fourth step of the Rivara algorithm.

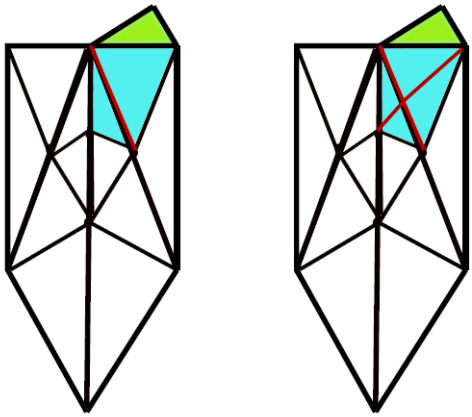


Figure 32: The fifth step of the Rivara algorithm.

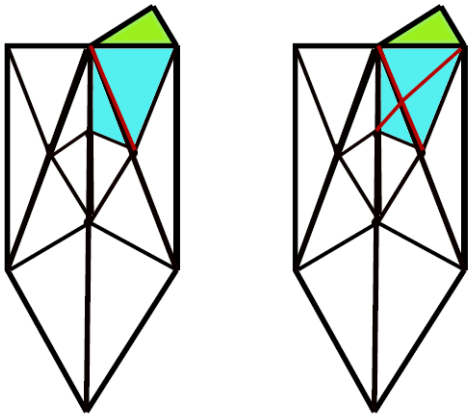


Figure 33: The sixth step of the Rivara algorithm.

step	*CHECK*	*BREAK*
1	4	2
2	3	2
3	5	2
4	4	2
5	3	2
6	2	2
total	21	12
total parallel	21	12

Table 1: The number of touched and split triangles in each step of the Rivara algorithm presented in Figures 28.

be reduced to 9 because we are splitting parallel triangles that do not share a broken edge.

Algorithm 3 Graph-grammar-based mesh refinement

Require: t_0 *triangle to break*, List *list of triangles to break*
1: Execute production **(P1)**(t_0)
2: Add to *List* neighbors broken edges of triangle t_0
3: **while** Non empty *List* **do**
4: RefList = List
5: Clear(List))
6: Execute production **(P2)** on triangles from RefList
7: **if** any triangle was broke **then**
8: Add neighbors of broken edges to *List*
9: continue
10: Execute production **(P3)** on triangles from RefList
11: **if** any triangle was broke **then**
12: Add neighbors of broken edges to *List*
13: continue
14: (similarly for other productions **(P4)**, **(P5)**, **(P6)**)
15: **end while**

Although it is impossible to derive a formula for the computational cost for a general mesh partitioned by classical and grammar-based algorithms, we estimated the costs on a representative example of the model. The classical longest edge refinement algorithm sequentially processes a single *LEPP*. Our graph grammar-based algorithm can check multiple triangles simultaneously and performs multiple breaks at the same time.

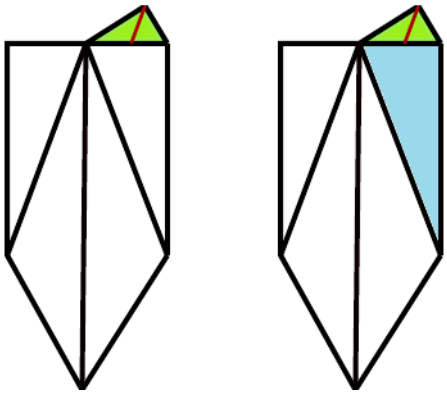


Figure 34: The first step of the graph-grammar-based algorithm.

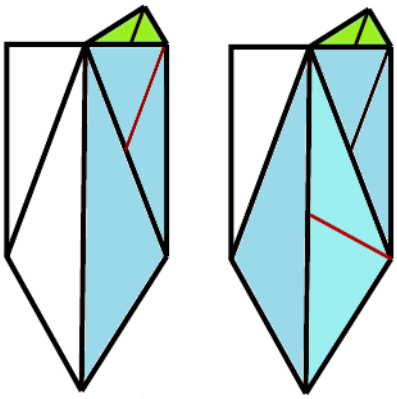


Figure 35: The second step of the graph-grammar-based algorithm.

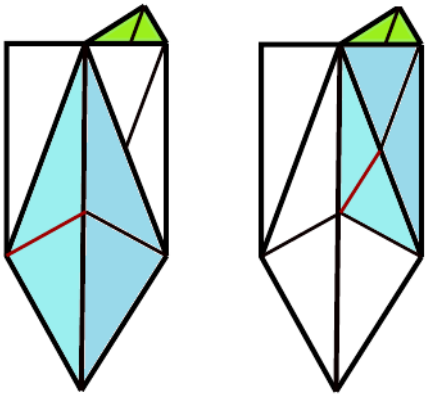


Figure 36: The third step of the graph-grammar-based algorithm.

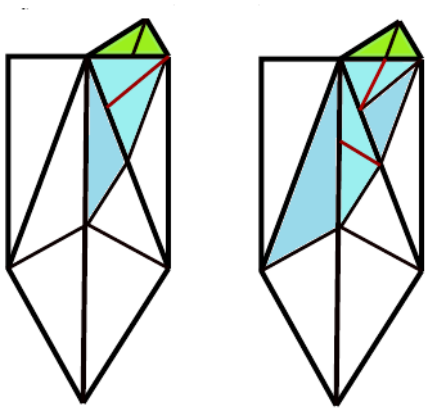


Figure 37: The fourth step of the graph-grammar-based algorithm.

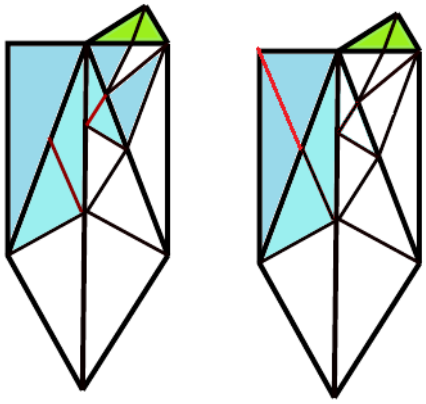


Figure 38: The fifth step of the graph-grammar-based algorithm.

step	*CHECK*s	*BREAK*s (P1)	(P2)	(P3)
1	1	1	0	0
2	1	0	0	1
3	5	0	0	1
4	5	0	1	0
5	4	0	1	0
6	4	0	0	1
7	5	0	1	1
8	8	0	1	1
9	7	0	1	1
10	4	0	1	0
total	44	1	5	6
total parallel	10		9	

Table 2: Number of trials (*CHECK*s) and applications (*BREAK*s) of particular productions executed by graph-grammar-based algorithm in nine steps presented in Figure 34.

3.3 MANUFACTURED SOLUTION ADVECTION-DIFFUSION PROBLEM

The goal of this section is to verify our solver by testing on a manufactured solution problem of advection-dominated diffusion. We choose the advection vector $\beta = (1,1)^T$ and $Pe = 1/\epsilon = 100$ and solve the advection-diffusion equation with homogeneous Dirichlet boundary conditions. We use the solution produced

$$u(x,y) = \left((1-x) + \frac{e^{Pe*(1-x)} - 1}{1 - e^{Pe}}\right)\left((1-y) + \frac{e^{Pe*(1-y)} - 1}{1 - e^{Pe}}\right)$$

enforced by the forcing term f. We set the reaction term to zero $c = 0$. This analytic solution expression restricts the number of Peclet to $Pe = 100$ due to machine precision. The solution has a stip gradient at left and bottom corners of the domain, where we expect the adaptations.

In Figures 39-51 we show the sequence of meshes generated by the adaptive algorithm. It captures the gradient at the left and bottom corners. Figure 52 shows the final mesh and the

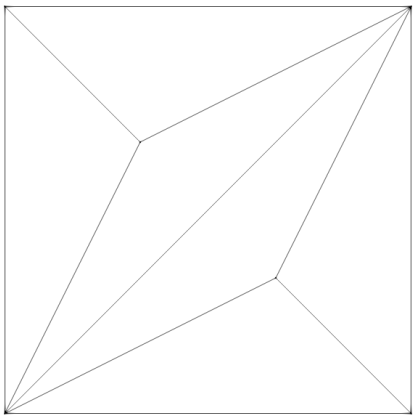


Figure 39: Sequence of adaptive meshes (1/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

Table 3: Convergence in L^2 norm.

iteration	1	2	3	4	5	6	7
L^2 norm	69.2	41.9	36.8	20.8	16.3	8.14	5.47
iteration	8	9	10	11	12	13	14
L^2 norm	2.24	1.09	0.38	0.12	0.03	0.009	0.003

final results. We also report the convergence of the L^2 norm in 3.

3.4 TOPOGRAPHIC MESH GENERATION IN LESSER POLAND AREA

In this section we run the graph-grammar-based longest edge refinement algorithm to generate the topographic mesh of the Lesser district of Poland. The mesh covers the topographic area of a size 42954.3 (+/- 446.3) x 3335 4.7 (+/- 574.8) meters. The initial mesh is not the exact rectangular since the Earth is not flat, and the input data are taken from the Earth database [16]. The initial mesh elements have vertices adjusted to the three-dimensional points located in the three-dimensional space, adjusted to the Earth's topography database.

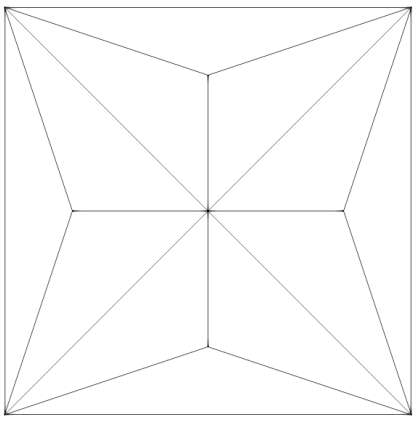


Figure 40: Sequence of adaptive meshes (2/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

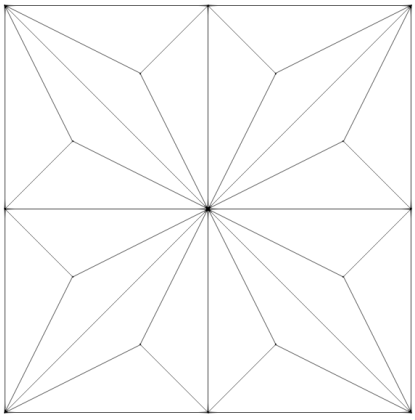


Figure 41: Sequence of adaptive meshes (3/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

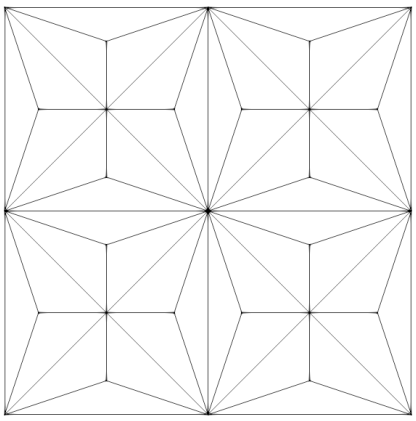


Figure 42: Sequence of adaptive meshes (4/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

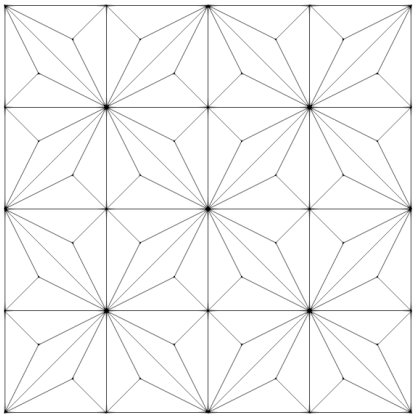


Figure 43: Sequence of adaptive meshes (5/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

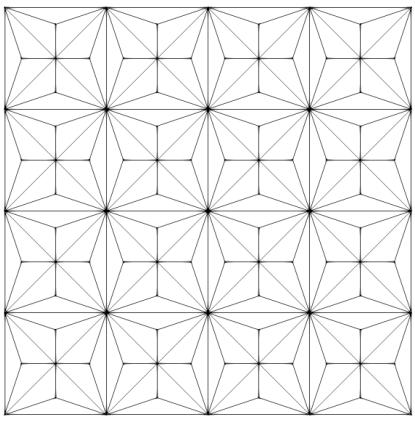


Figure 44: Sequence of adaptive meshes (6/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

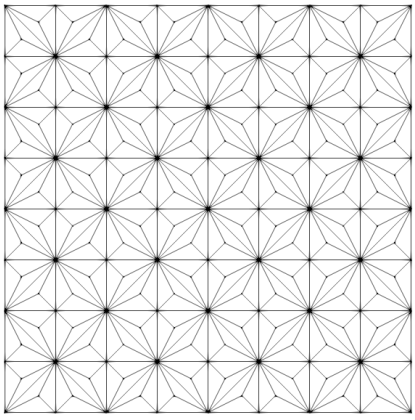


Figure 45: Sequence of adaptive meshes (7/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

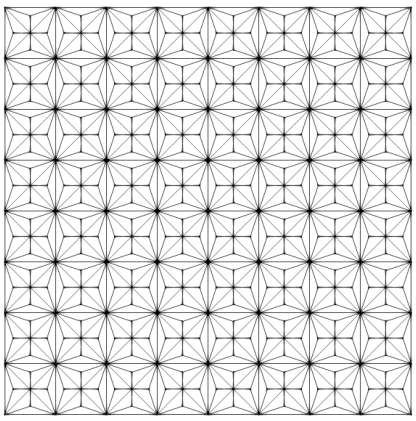


Figure 46: Sequence of adaptive meshes (8/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

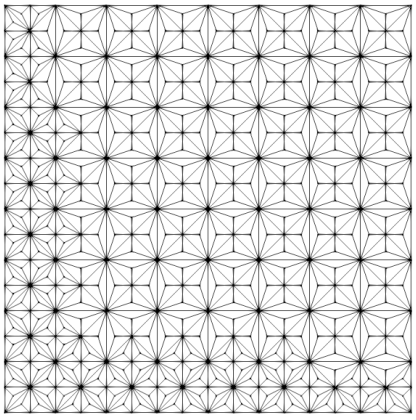


Figure 47: Sequence of adaptive meshes (9/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

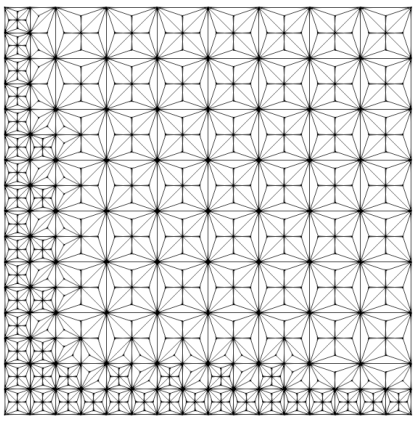


Figure 48: Sequence of adaptive meshes (10/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

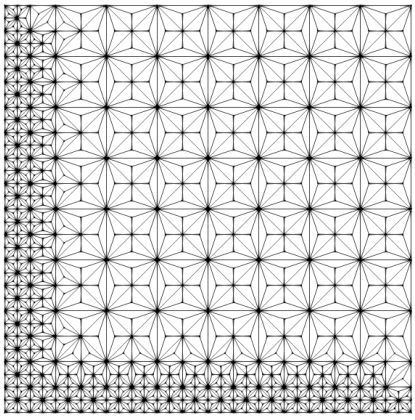


Figure 49: Sequence of adaptive meshes (11/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

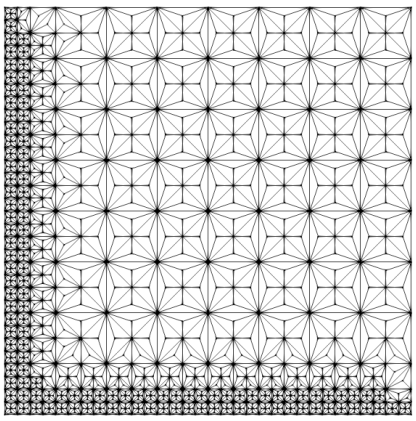


Figure 50: Sequence of adaptive meshes (12/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

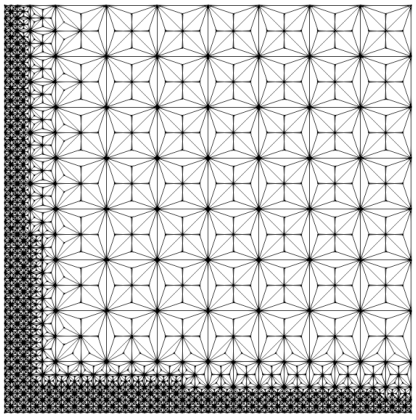


Figure 51: Sequence of adaptive meshes (13/13) generated by the graph-grammar-based solver for the problem of manufactured solution of advection-diffusion equations stabilized with SUPG.

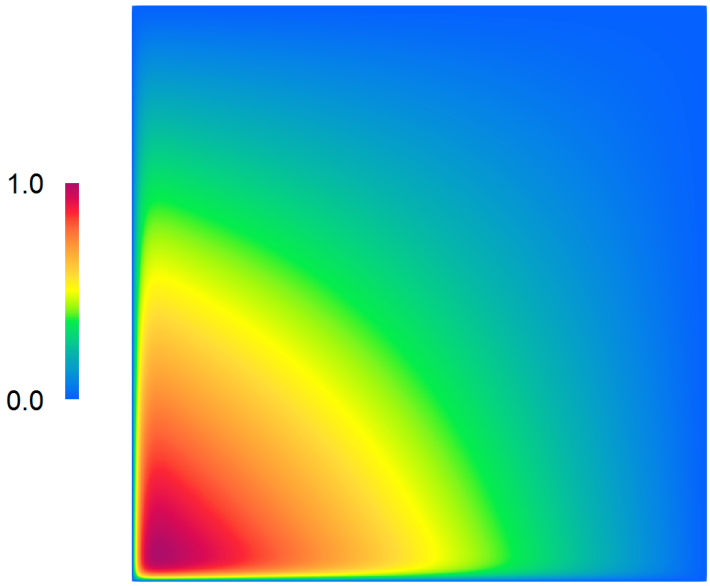


Figure 52: Result to the manufactured solution problem computed on the final mesh.

We run 24 iterations of our algorithm. Figures 56-65 presents snapshots from the refinement process. The final triangulation of the topographic surface of the Lesser district of Poland is presented in Fig. 65.

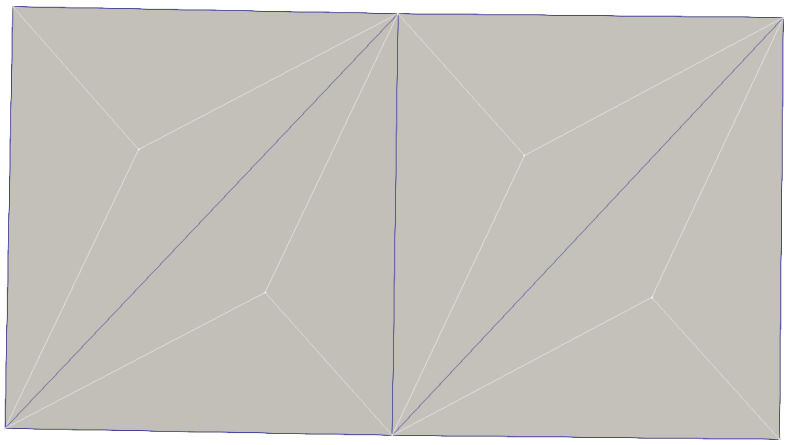


Figure 53: Snapshots (1/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

3.5 POLLUTION SIMULATIONS IN LESSER POLAND AREA

In this section we present a pollution simulations in Lesser District of Poland. We generate a three-dimensional grid on a two-dimensional grid with four layers of prisms, each divided into three tetrahedrons. We first focus on computing the wind dis-

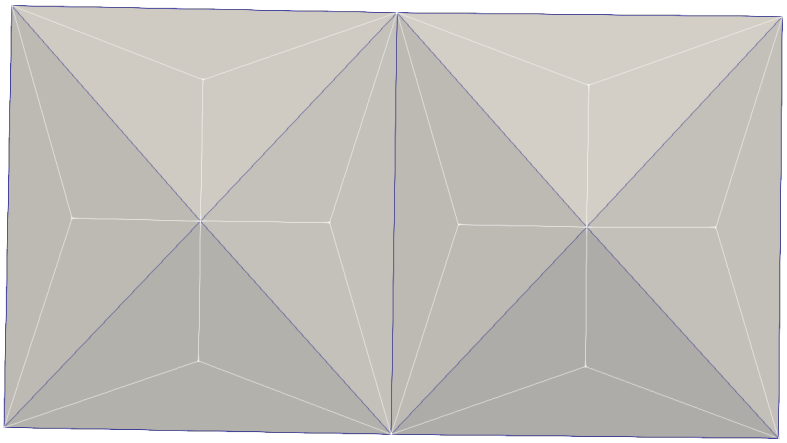


Figure 54: Snapshots (2/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

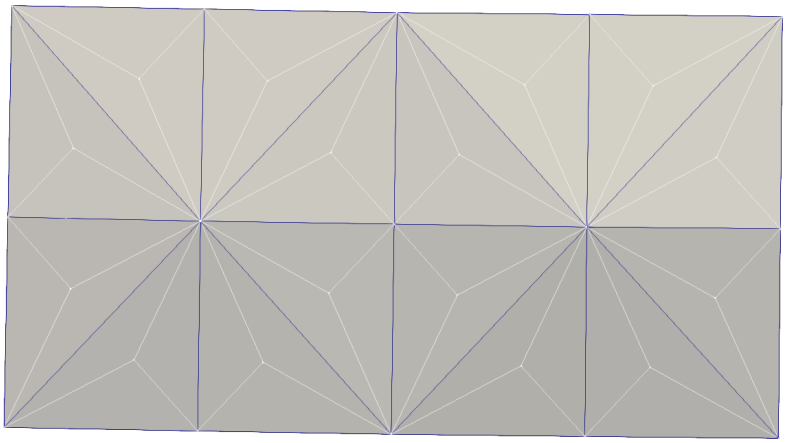


Figure 55: Snapshots (3/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

tribution over the entire domain based on two constant velocity values from actual measurements from a station on Kasprowy Wierch and a station in Zakopane. We generalize these measurements to the entire domain by solving the $\mathrm{div}\, u = 0$. The results are shown in Figures 66-68. We have a slightly time-varying northwest wind.

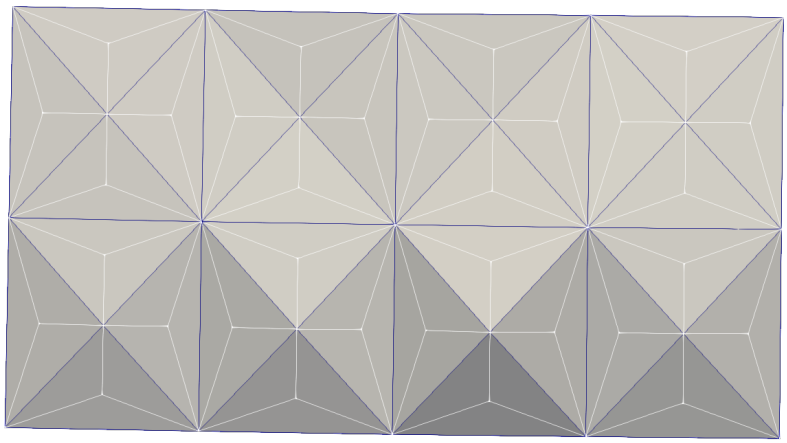


Figure 56: Snapshots (4/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

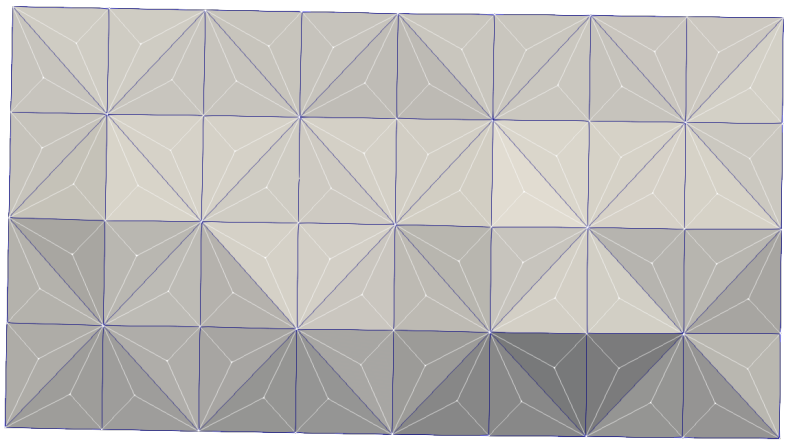


Figure 57: Snapshots (5/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

Having the wind field, $\beta(x, y, z, t)$ representing the north-western wind, we plug it as an advection field to the advection-diffusion-reaction problem

$$\frac{\partial u}{\partial t} + \beta \cdot \nabla u - \nabla \cdot (K \nabla u) = s(u) \tag{51}$$

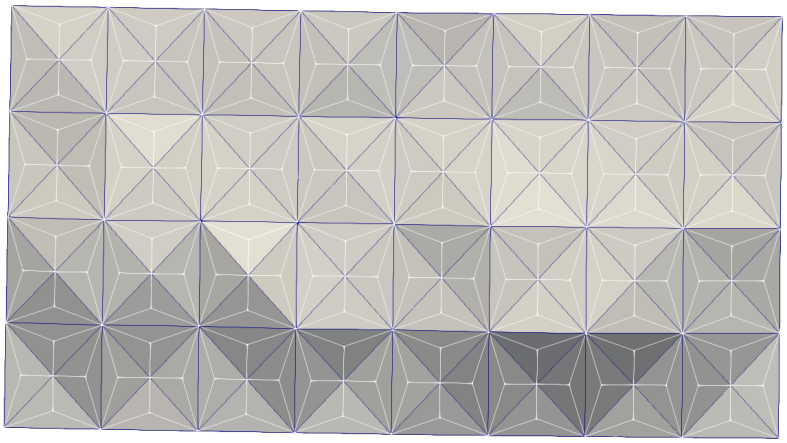


Figure 58: Snapshots (6/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

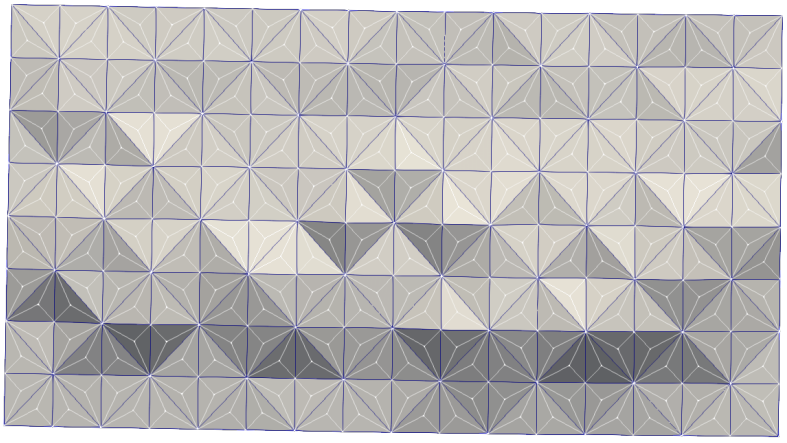


Figure 59: Snapshots (7/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

Here $u(x, y, z, t)$ is the unknown concentrations of the pollution in the area. In this simulation, since we have one scalar field

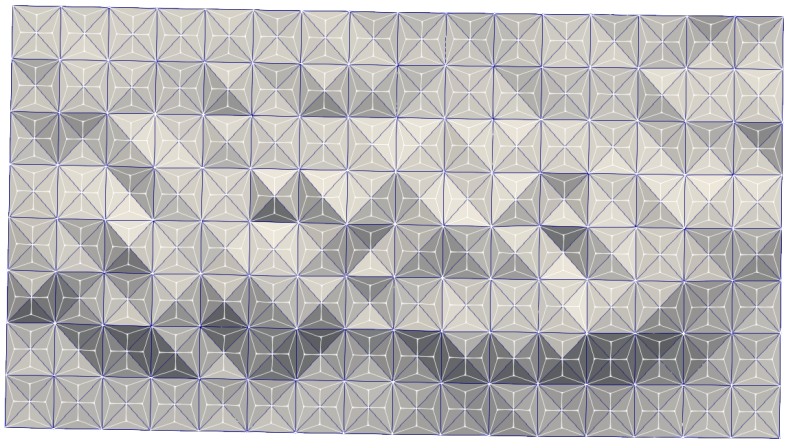


Figure 60: Snapshots (8/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

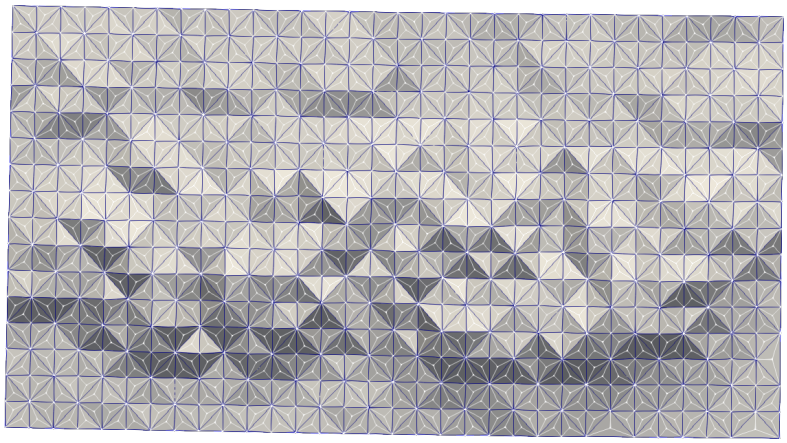


Figure 61: Snapshots (9/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

representing the pollution, we ignore the reaction part. Here K is the diagonal diffusion matrix

$$K = \begin{bmatrix} 8 * 10^{-6} & 0 & 0 \\ 0 & 8 * 10^{-6} & 0 \\ 0 & 0 & 4 * 10^{-6} \end{bmatrix} m^2/s \tag{52}$$

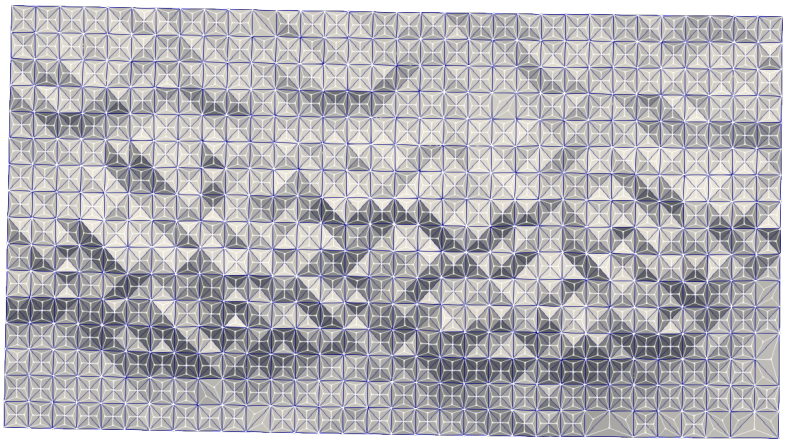


Figure 62: Snapshots (10/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

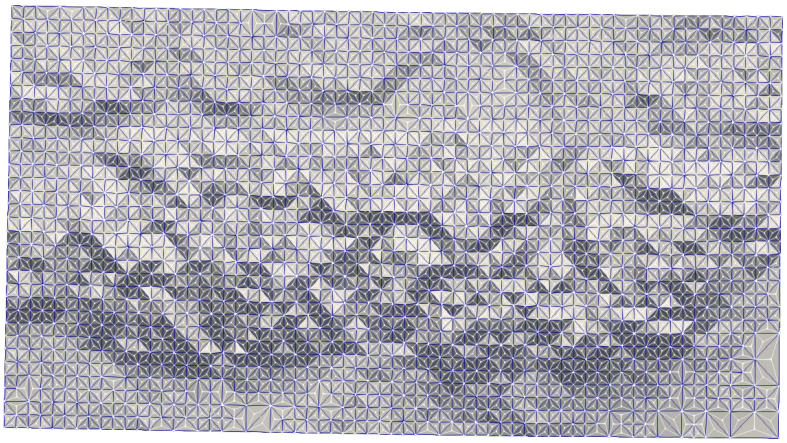


Figure 63: Snapshots (11/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

where the horizontal diffusion coefficient $8 * 10^{-6} m^2/s$, the vertical diffusion coefficient is $4 * 10^{-6} m^2/s$. We assume that the pollution comes from the north boundary and is blown inside the domain from the north boundary by the northwest wind calculated from actual measurements.

$$u = \beta, \tag{53}$$

and we have

$$u(x, y, z, t) = 0, \tag{54}$$

at the wind outlet. Moreover, we have

$$n \cdot (K \nabla u) = -V^d u \tag{55}$$

at the terrain level, where $V^d = 1.3 * 10^{-3}$ m/s.

We run the entire simulation on ATARI Linux cluster node We present snapshots from the simulation in Figures 69-98. They present the propagation of the pollution field, starting from the north boundary of the domain, going over the terrain as propagated by the north-western wind.

3.6 PERFORMANCE OF GRAPH-GRAMMAR BASED PARALLEL IMPLEMENTATION

We implemented our graph grammar-based system in the GALOIS framework developed by the group of prof. Keshav Pin-

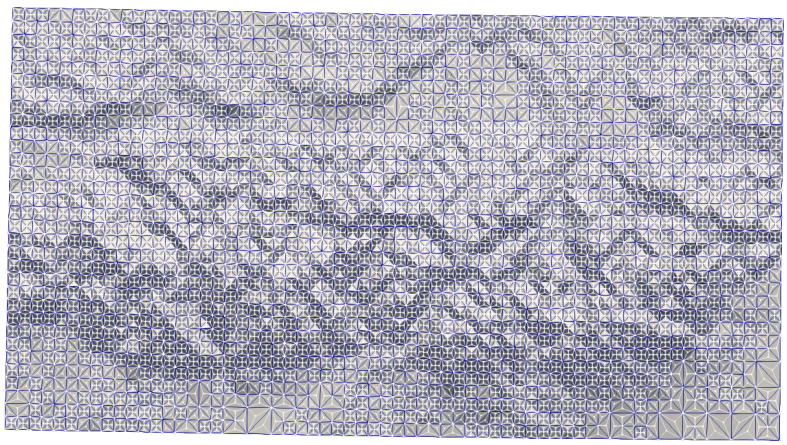


Figure 64: Snapshots (12/12) from the sequence of meshes generated by the graph-grammar-based longest-edge refinement algorithm.

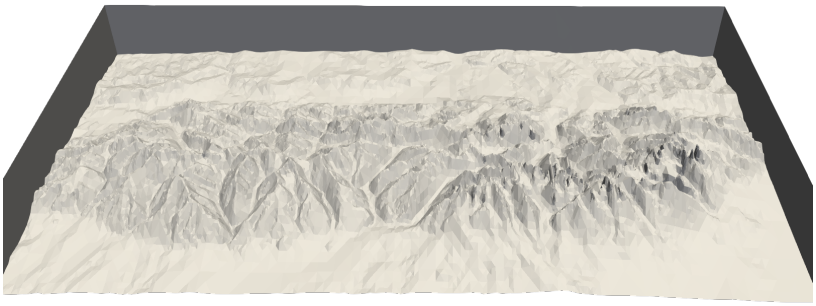


Figure 65: Final mesh generated by the graph-grammar-based longest-edge refinement algorithm representing Lesser District of Poland (South of Poland).

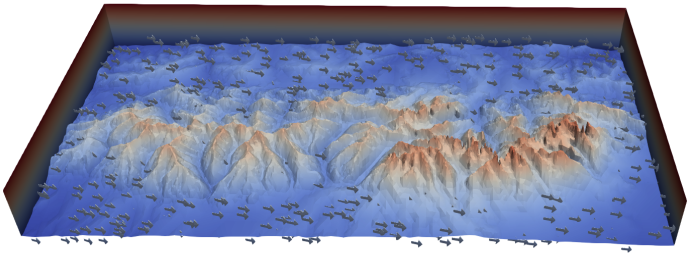


Figure 66: The wind vector field computed by solving $\mathrm{div} u = 0$ with the equations modified according to two-measurement stations. One snapshot of the wind field, at the beginning of the simulation. The maximum wind speed varies between 33 and 49 km/s.

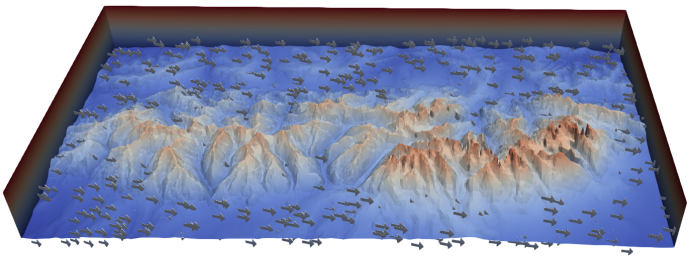


Figure 67: The wind vector field computed by solving $\mathrm{div} u = 0$ with the equations modified according to two-measurement stations. One snapshot of the wind field, in the midle of the simulation. The maximum wind speed varies between 33 and 49 km/s.

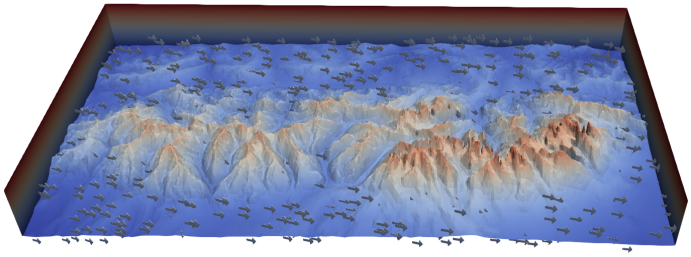


Figure 68: The wind vector field computed by solving divu = 0 with the equations modified according to two-measurement stations. One snapshot of the wind field, at the end of the simulation. The maximum wind speed varies between 33 and 49 km/s.

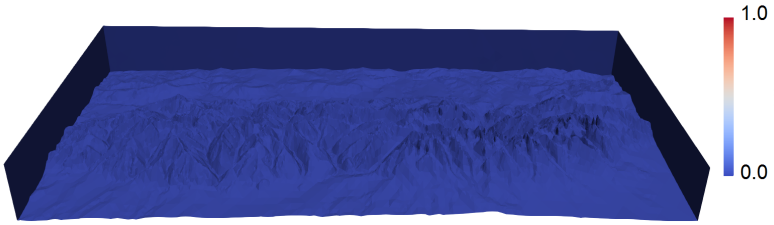


Figure 69: Pollution scalar field propagated by the north-western wind, front view (1/10).

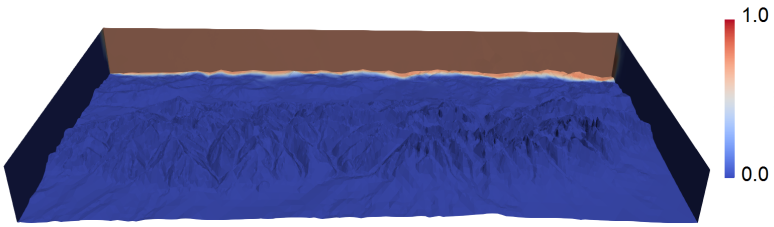


Figure 70: Pollution scalar field propagated by the north-western wind, front view (2/10).

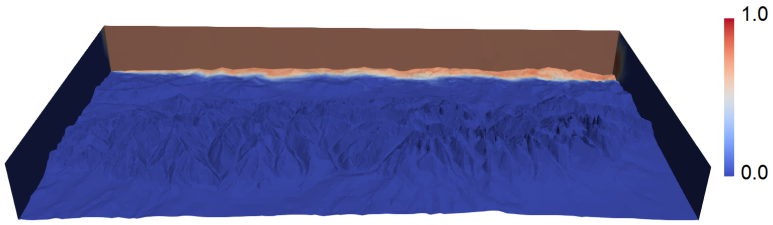


Figure 71: Pollution scalar field propagated by the north-western wind, front view (3/10).

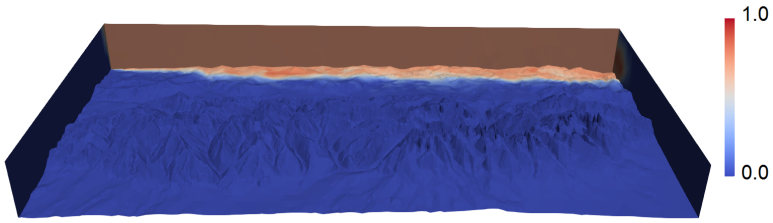


Figure 72: Pollution scalar field propagated by the north-western wind, front view (4/10).

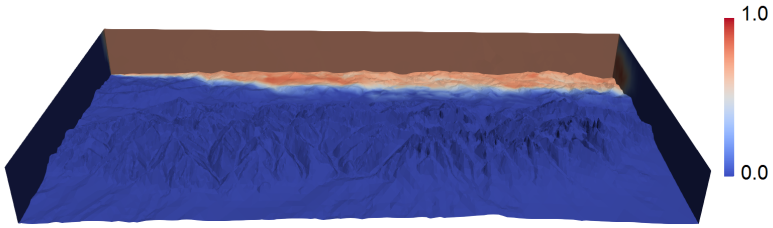


Figure 73: Pollution scalar field propagated by the north-western wind, front view (5/10).

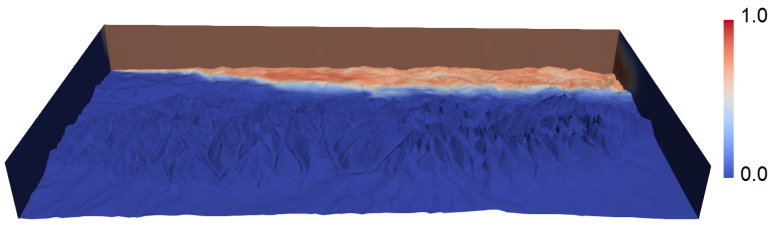


Figure 74: Pollution scalar field propagated by the north-western wind, front view (6/10).

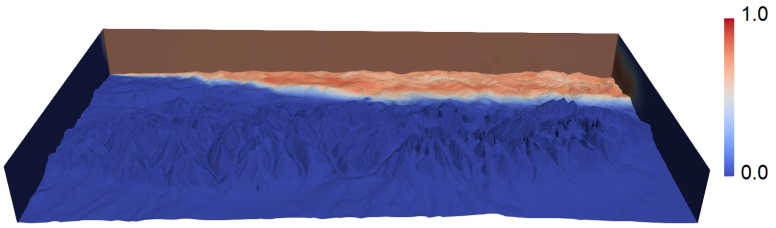


Figure 75: Pollution scalar field propagated by the north-western wind, front view (7/10).

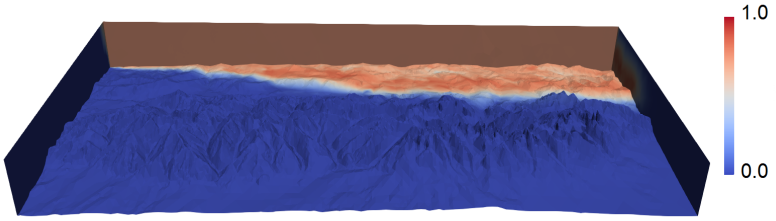


Figure 76: Pollution scalar field propagated by the north-western wind, front view (8/10).

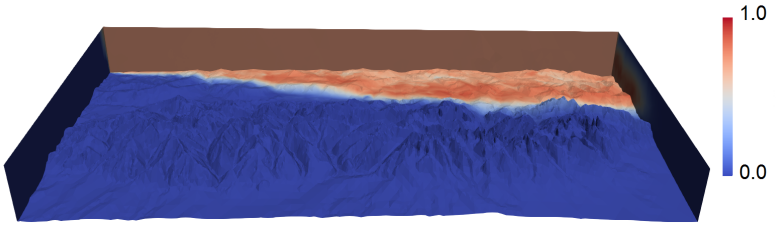


Figure 77: Pollution scalar field propagated by the north-western wind, front view (9/10).

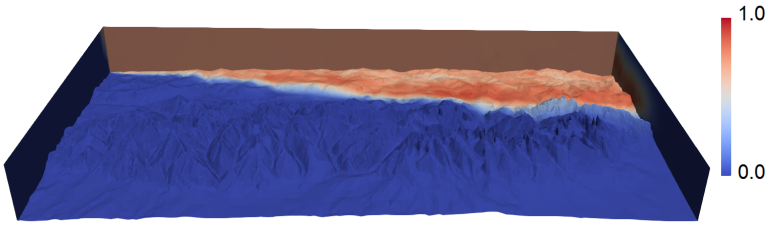


Figure 78: Pollution scalar field propagated by the north-western wind, front view (10/10).

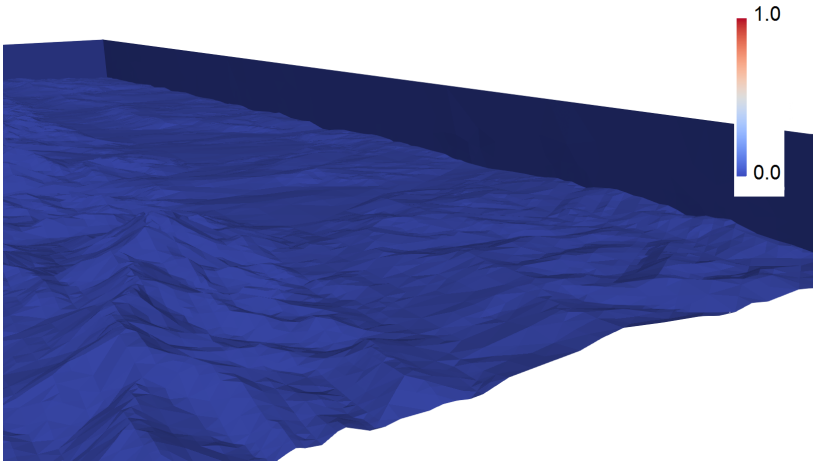


Figure 79: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (1/10).

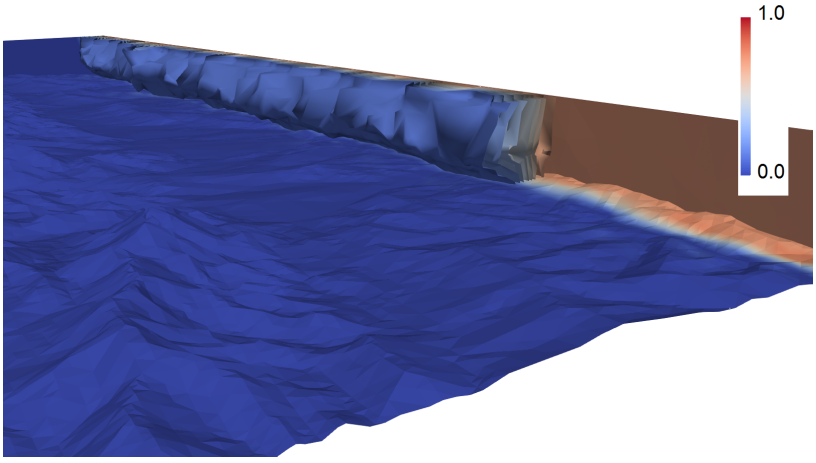


Figure 80: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (2/10).

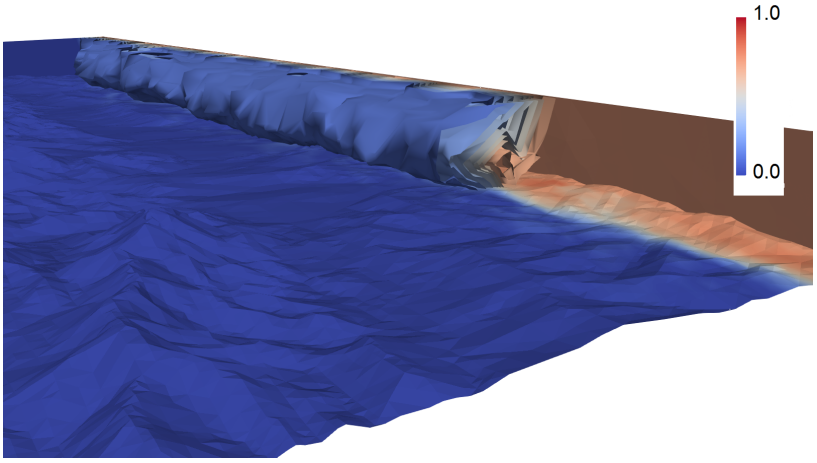


Figure 81: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (3/10).

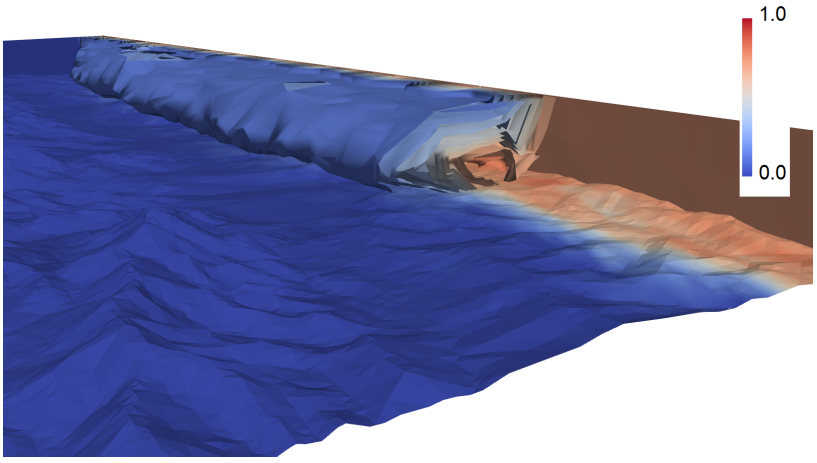


Figure 82: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (4/10).

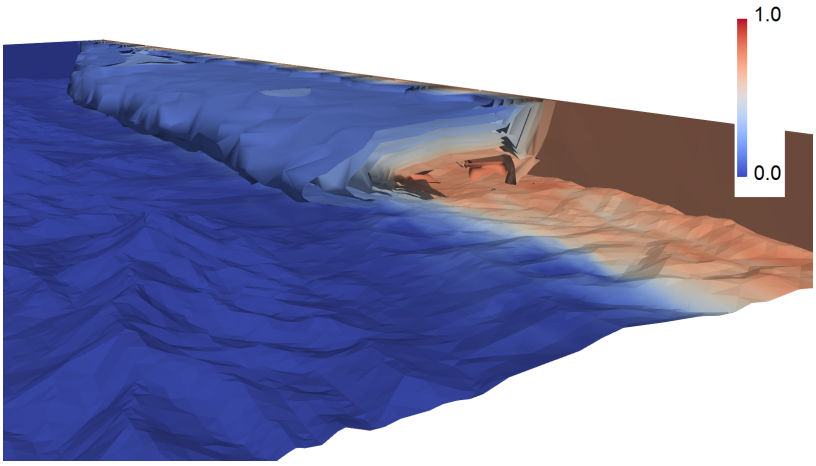


Figure 83: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (5/10).

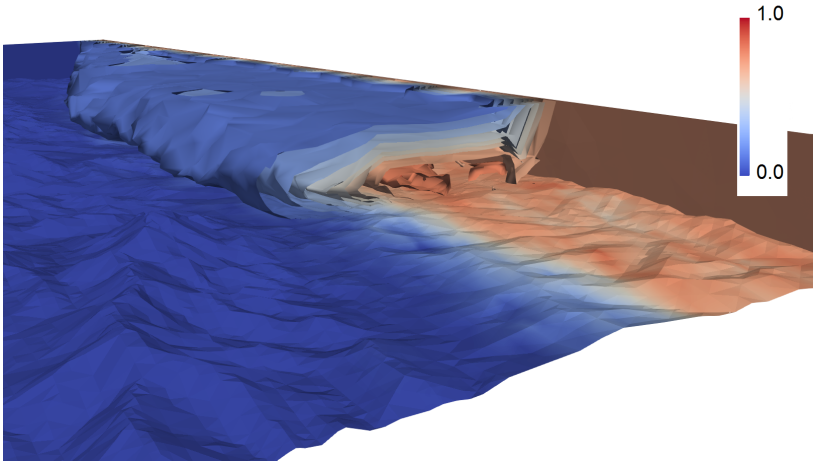


Figure 84: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (6/10).

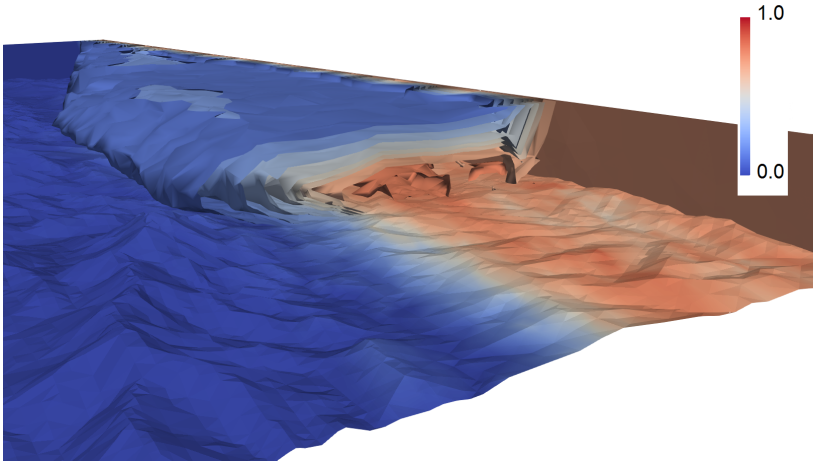


Figure 85: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (7/10).

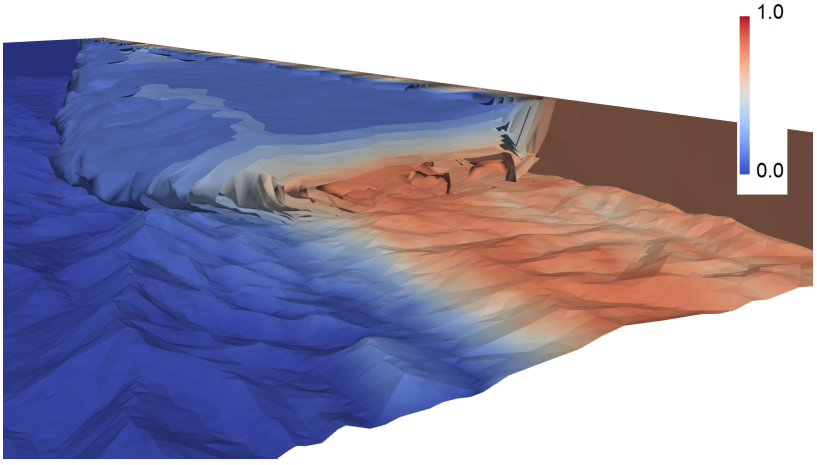


Figure 86: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (8/10).

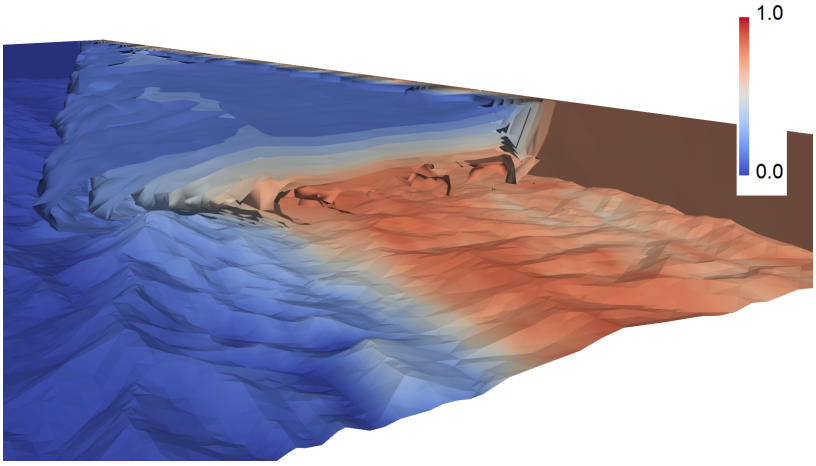


Figure 87: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (9/10).

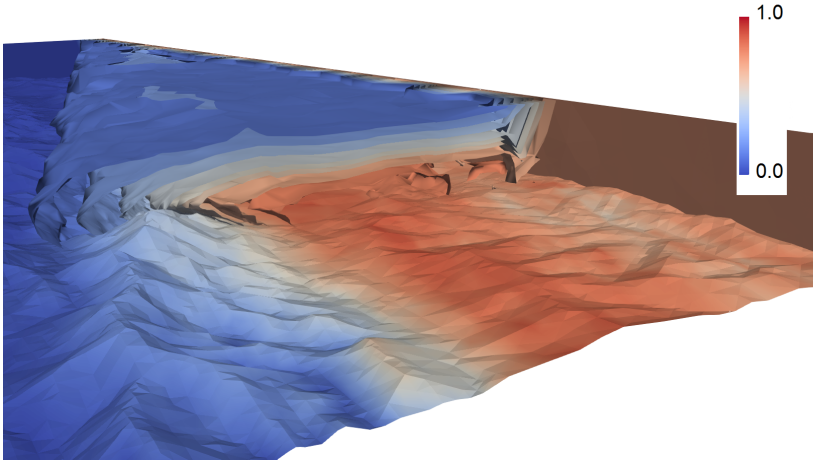


Figure 88: Pollution scalar field propagated by the north-western wind, zoom towards right top corner, and plotting the cross-section and contours (10/10).

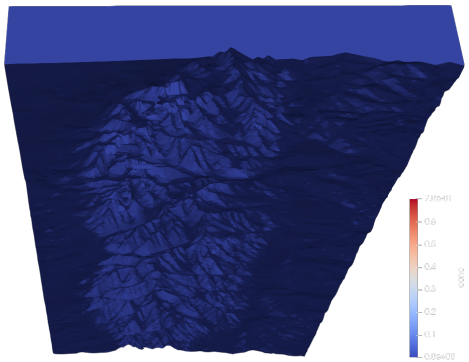


Figure 89: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (1/10).

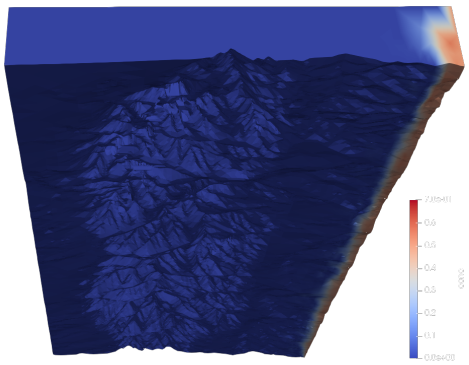


Figure 90: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (2/10).

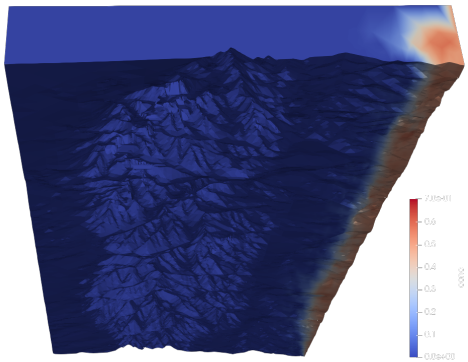


Figure 91: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (3/10).

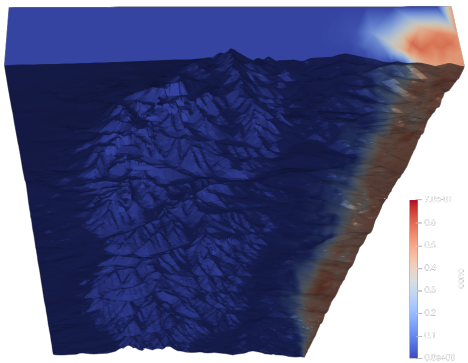


Figure 92: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (4/10).

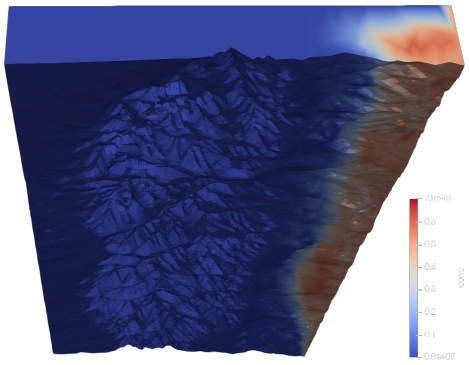


Figure 93: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (5/10).

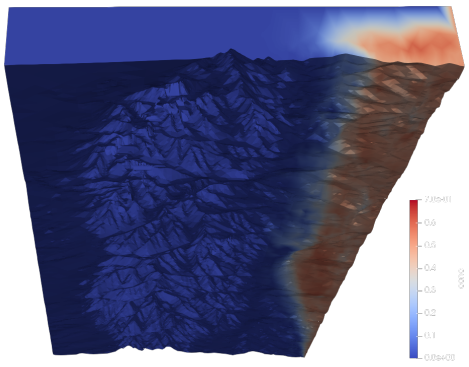


Figure 94: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (6/10).

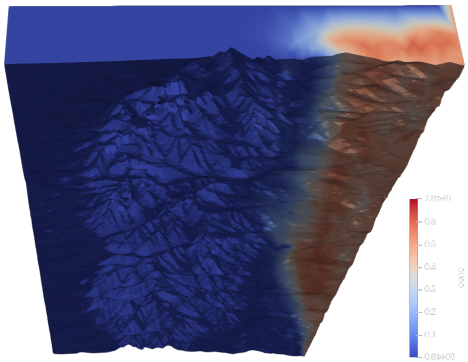


Figure 95: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (7/10).

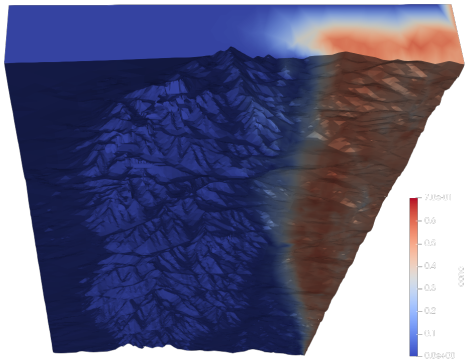


Figure 96: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (8/10).

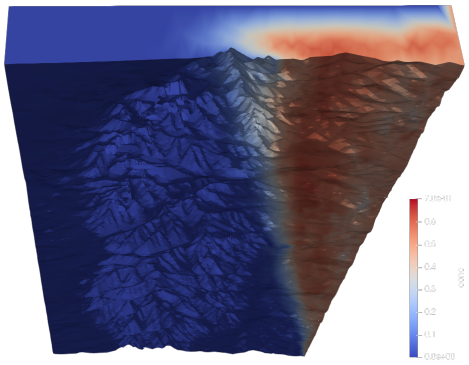


Figure 97: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (9/10).

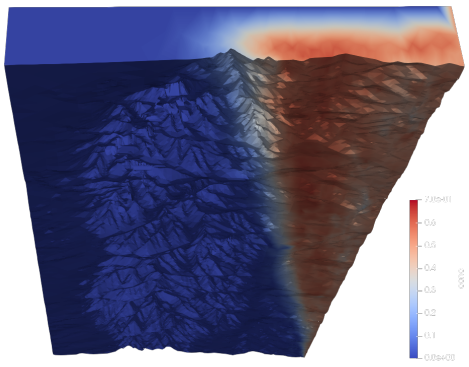


Figure 98: Pollution scalar field propagated by the north-western wind, view from "under the ground-level" (10/10).

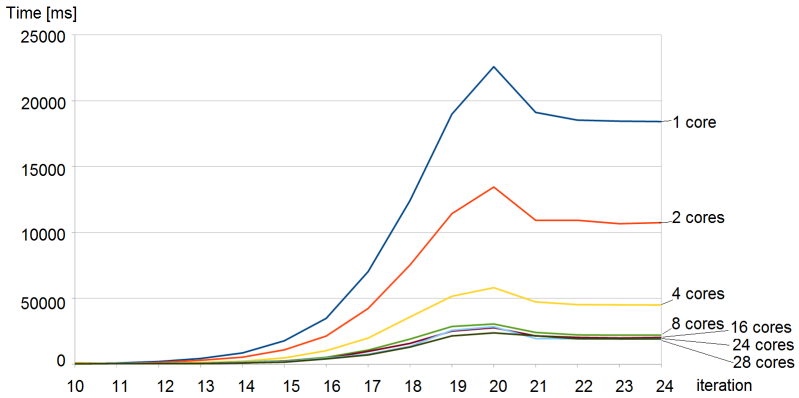


Figure 99: Execution times on the last 15 iterations.

gali [20, 66], which enables concurrent graph processing. The code is compiled on node 13 on an Atari Linux cluster from the Adaptive Algorithms and Systems research group (a2s.agh.edu.pl) from the AGH Institute of Computer Science. Node 13 has an Intel(R) Xeon(R) CPU E5-2680 v4 with a clock speed of 2.40 GHz and 28 cores. The code requires gcc/8.1, boost, and cmake. The code can be downloaded from *https://github.com/Podsiadlo/Terrain MeshGenerator/ Lonestar/graphgrammar2*. Here we present scalability results for the concurrent code for 24 iterations. It converts the initial mesh of twenty four triangles into a final mesh with more than 10^7 triangles. In Table *4* and Fig. *99* we report the execution time of the last fifteen iterations of the mesh generation algorithm, with the number of cores used increasing from 1 to 28. In Table *5* and Fig. *100* we report the speedup for the last fifteen iterations of the mesh generation algorithm, with the number of cores used increasing from 1 to 28. We report times and acceleration for iterations 15-24. Previous iterations took less than 100 milliseconds, so we skip them from our analysis. Note that once the assumed accuracy of the terrain approximation is reached at step twenty, the computational cost per step decreases because we are not performing massive refinements on the entire grid.

Table 4: Execution times on 28 cores on Atari Linux cluster node, for the iterations 10-24. The previous iterations took less than 100 milliseconds.

cores	1	2	4	8	16	24
step10	51	52	115	51	51	25
step11	92	59	52	52	49	73
step12	208	135	77	51	64	61
step13	434	298	131	61	52	51
step14	855	535	226	177	88	155
step15	1780	1091	479	250	220	198
step16	3482	2140	1026	524	438	454
step17	7031	4232	1984	1072	974	762
step18	12425	7536	3588	1907	1581	1354
step19	18985	11427	5156	2865	2512	2578
step20	22582	13450	5807	3052	2768	2848
step21	19113	10923	4726	2411	2149	1947
step22	18527	10925	4521	2226	2034	1929
step23	18451	10664	4504	2204	1992	1908
step24	18418	10746	4494	2212	2031	1941
Total:	144s	85s	38s	21s	18s	17s

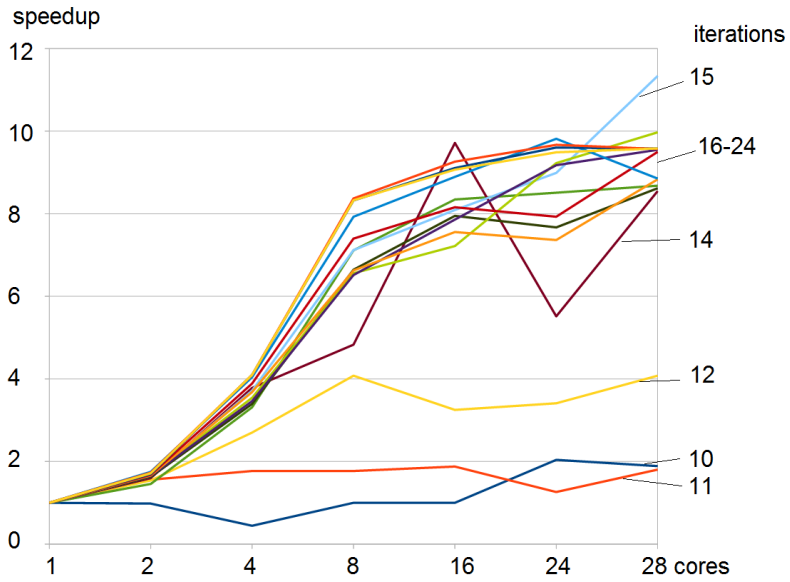


Figure 100: Speedup on the last 3 iterations of mesh generation process.

Table 5: Speedup up to 28 cores on Atari Linux cluster node, for iterations 10-24. The previous iterations took in total less than 100 milliseconds.

cores	1	2	4	8	16	24
step10	1	0,98	0,44	1,00	1,00	2,04
step11	1	1,56	1,77	1,77	1,88	1,26
step12	1	1,54	2,70	4,08	3,25	3,41
step13	1	1,46	3,31	7,11	8,35	8,51
step14	1	1,60	3,78	4,83	9,72	5,52
step15	1	1,63	3,72	7,12	8,09	8,99
step16	1	1,63	3,39	6,65	7,95	7,67
step17	1	1,66	3,54	6,56	7,22	9,23
step18	1	1,65	3,46	6,52	7,86	9,18
step19	1	1,66	3,68	6,63	7,56	7,36
step20	1	1,68	3,89	7,40	8,16	7,93
step21	1	1,75	4,04	7,93	8,89	9,82
step22	1	1,70	4,10	8,32	9,11	9,60
step23	1	1,73	4,10	8,37	9,26	9,67
step24	1	1,71	4,10	8,33	9,07	9,49

Subdomains=Processors	Grid Grid	50x50x50 Time [s]	50x50x100 Time [s]
1	(1,1,1)	19	-
2	(1,1,2)	23	-
4	(1,1,4)	23	120
8	(2,1,4)	63	157
16	(2,2,4)	36	152
32	(4,2,4)	42	150
64	(4,4,4)	49	157
128	(8,4,4)	63	160
256	(8,8,4)	72	166
512	(16,8,4)	-	170
1024	(16,16,4)	-	178

Table 6: Weak scallability of the alternating-directions solver up to 1024 processors (subdomains), partitioned into different parts of the parallel solver from [63].

3.7 SCALABILITY OF THE PARALLEL ALTERNATING-DIRECTIONS SOLVER OVER NON-REGULAR TERRAIN TOPOGRAPHY

This section presents scalability results for the alternating-directions solver performed on irregular terrain geometry.

The method for incorporating irregular topography into a directional split solver is implemented in fortran95 code developed in collaboration with Petar Minev's group [63]. The parallelization of this code was not part of this dissertation. The code uses OpenMP and MPI libraries for parallelization. It does not use any other libraries and it is a highly optimized code. In the figure 101 and in table 6 we report the poor scalability for two different subdomain sizes: 50x50x100, and 50x50x50. For numerical verification of the code, we refer to [5]. In Figure 102 we show some snapshots from preliminary simulations. Here we focus on verifying the scalability of the code, leaving model formulation and large-scale massively parallel simulations of various atmospheric phenomena for the future. This in itself will be a challenging task, requiring reliable data for the initial state, forcing, and boundary conditions.

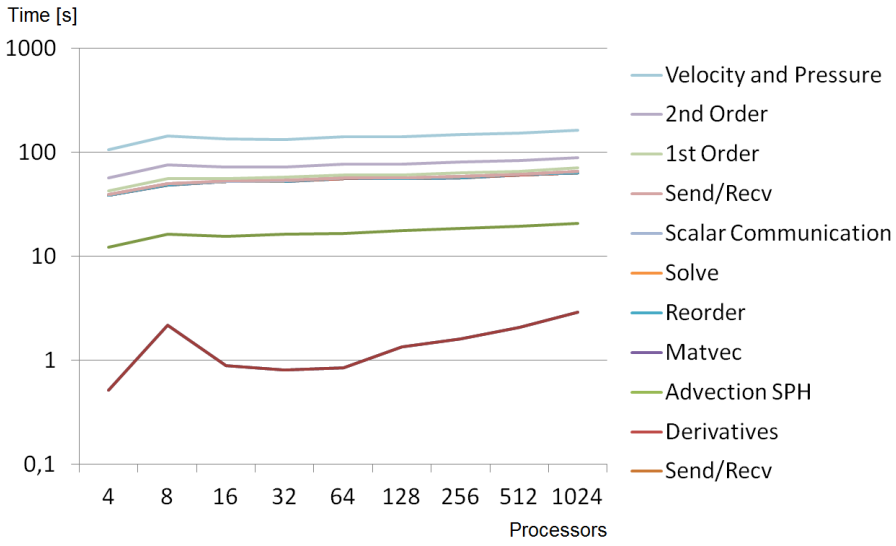


Figure 101: Weak scallability of the alternating-directions solver for subdomains with with $49 \times 49 \times 99$, and $49 \times 49 \times 49$ internal points, one subdomain per processor, up to 1024 processors (subdomains)

3.7.1 *Comparison of alternating-directions and graph-grammar solvers*

- The alternating directions solver does not allow for mesh adaptation. To provide a high accuracy solution it generates an uniform mesh with small elements. Let us compare this solver to the graph-grammar based solver that requires 24 adaptive iterations in some areas. The uniform mesh employed by the alternating-directions solver to approximate the terrain with atmospheric phenomena in an accurate way requires elements with size similar to the smallest adaptive elements. The size of the smallest mesh element is $\frac{1}{2^{24}} = 0.000000059604644775390625 \approx 0.00000006 = 6 * 10^{-8}$. The regular mesh employed by the alternating directions solver requires $10^8 \times 10^8 = 10^{16}$ two-dimensional mesh elements covering the terrain to deliver similar accuracy than graph-grammar based adaptive solver in this area.

- The alternating-directions solver can process $800 \times 800 \times 200$ mesh points using 1024 processors, which is equivalent to 9 adaptive iterations of the graph-grammar solver. For larger grids it runs out of memory, or it requires special access to larger resources at the supercomputing cen-

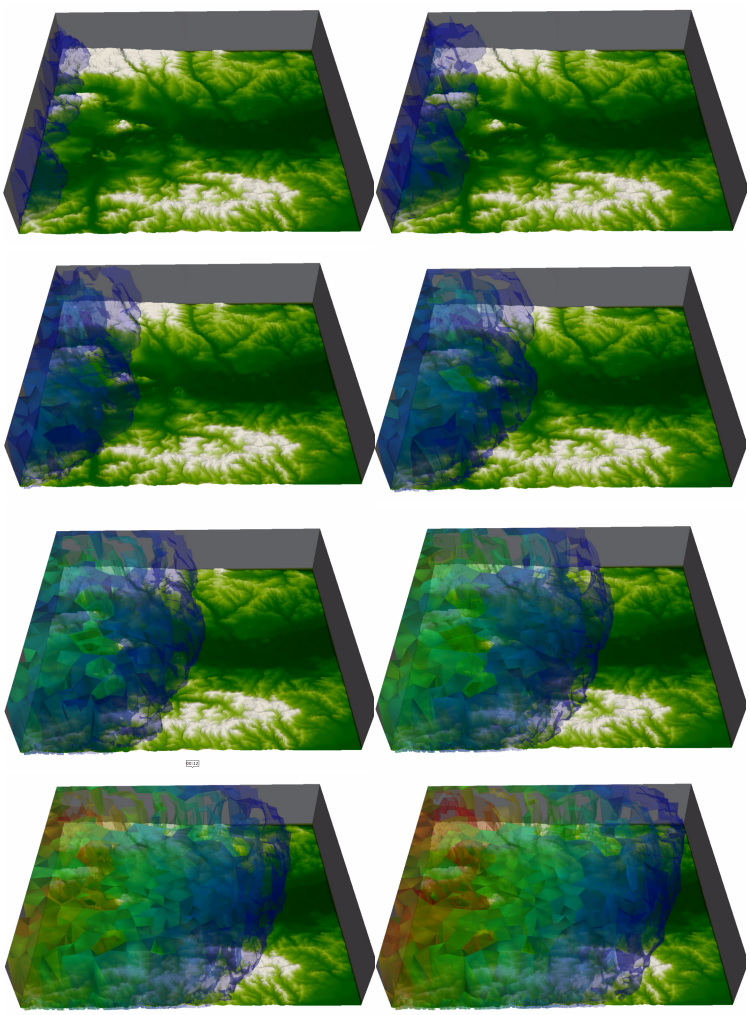


Figure 102: Snapshots from the simulation

ter. Our graph-grammar solver can perform 24 adaptive iterations on a single laptop, going down to the element size $\frac{1}{2^{24}}$ in the areas that requires special attention.

- The matrix-free iterative solver using the element matrix multiplication by vector followed by an assembly to the right-hand side allows for efficient execution on a single laptop, without a necessity of using the large super-computing centers. The execution time of the entire simulation of the pollution propagation in Lesser district of Poland with the adaptive graph-grammar based code generating local element sizes of 10^{-8} takes less than 1 hour on a single workstation with 24 cores.

Part IV

CONCLUSIONS AND FUTURE WORK

4

CONCLUSIONS AND FUTURE WORK

This thesis shows how to express by graph-grammar productions the longest-edge mesh refinement algorithm for a two-dimensional mesh with triangular elements. The graph-grammar-based algorithm allows for better parallelization than classical Rivara's algorithm. We also show how to extend it to the three-dimensional grids and interface with GMRES solver and Crank-Nicolson time integration scheme. The mesh generation algorithm removes all the hanging nodes automatically from the mesh. The stabilized advection-diffusion-reaction solver executed on the computational mesh based on topographic data of Lesser Poland area provides a tool for the pollution propagation simulations.

The main scientiffic achievement of this thesis can be summarized as follows:

- We designed and implemented an algorithm allowing for pollution simulations over complicated terrain topography, with adaptive finite element method and graph grammar based longest-edge refinement. We employed advection-diffusion-reaction model for simulation of the pollution propagation over Lesser district of Poland.
- We showed that the graph-grammar based implementation of the longest edge refinement algorithm allows for additional parallelization within a single longest-edge refinement path, thus allowing for extra speedup.
- We employed the graph-grammar based longest-edge refinement algorithm for modeling the terrain topography and we extend it to model the generation of the three-dimensional tetrahedral meshes span over the terrain mesh.
- We incorporated the graph grammar model with the finite element method stabilized with Streamling-Upwind-Petrov-Galerkin (SUPG) method for the non-stationary advection-diffusion-reaction simulations.
- We perform parallel pollution simulations in Lesser Poland area.

- We showed how to incorporate non-regular terrain topography into the alternating direction solver of Navier-Stokes-Boussinesq equations with finite difference method.
- We verified the linear computational cost of the alternating direction solver on parallel Linux cluster

We also proposed a method to include non-regular topographic data into alternating-direction solver with finite difference method. We compared the graph-grammar based solver with alternating-directions solver. We showed that despite linear computational cost, the alternating direction solver in its current form does not allow for mesh adaptation, which makes the graph-grammar based solver more attractive.

The future work may include

- the graph-grammar based simulations of the pollution resulting from point sources in the Kraków city area, e.g., from the factories' chimneys;
- expression of the direction splitting algorithm simulations by graph-grammar productions;
- incorporation of the thermal inversion effects simulated with Navier-Stokes-Boussinesq equations [63] into the graph-grammar based solver;
- research on a possibility of incorporation of the adaptive algorithms into alternating-direction solver.

BIBLIOGRAPHY

[1] Philip Hauge Abelson. "Air Pollution and Acid Rain." In: *Science* 230.4726 (1985), pp. 617–617. DOI: 10.1126/science.230.4726.617.

[2] European Environment Agency. *Air Quality in Europe - 2017 report*. 2017.

[3] Nash'at N. Ahmad, David P. Bacon, Mary S. Hall, and Ananthakrishna Sarma. "Application of the multidimensional positive definite advection transport algorithm (MPDATA) to environmental modelling on adaptive unstructured grids." In: *International Journal for Numerical Methods in Fluids* 50.10 (2006), pp. 1247–1268. DOI: 10.1002/fld.1113.

[4] "Applying Cooperating Distributed Graph Grammars in Computer Aided Design." In: *Lecture Notes in Computer Science* 3911 (2005), pp. 567–574.

[5] Petar Minev Aziz Takhirov Roman Frolov. "Direction splitting scheme for Navier-Stokes-Boussinesq system in spherical shell geometries." In: *arXiv:1905.02300* (2019).

[6] Marian Bubak, Jacek Kitowski, and Kazimierz Wiatr. "E-Science on Distributed Computing infrastructure." In: *Achievements of PL-Grid Plus Domain-specific Services and Tools* 8500 (2014).

[7] Graham F. Carey and John Tinsley Oden. "Finite Elements: Computational Aspects." In: (1984).

[8] Erzsébet Csuhaj-Varjú. "Grammar systems: a short survey." In: *Grammar Systems Week 2004, Budapest. Proceedings.* Ed. by E Csuhaj-Varjú and GY Vaszil. Budapest: MTA SZTAKI, 2004, pp. 141–157. URL: http://eprints.sztaki.hu/3684/.

[9] Erzsébet Csuhaj-Varjú, J. Dassow, Jozef Kelemen, and Gh. Paun. "Grammar systems. A grammatical approach to distribution and cooperation." In: *Topics in Computer Mathematics* 8 (1994).

[10] Erzsébet Csuhaj-Varjú, J. Dassow, and Gh. Paun. "Dynamically controlled cooperating/distributed grammar systems." In: *Information Sciences* 69.1-2 (1993), pp. 1–25.

[11] Erzsébet Csuhaj-Varjú and G. Vaszil. "On context-free parallel communicating grammar systems: Synchronization, communication, and normal forms." In: *Theoretical Computer Science* 255.1-2 (2001), pp. 511–538.

[12] Leszek Demkowicz. *Computing with hp-adaptive finite elements: volume 1 one and two dimensional elliptic and Maxwell problems*. CRC press, 2006.

[13] Leszek Demkowicz, Jason Kurtz, David Pardo, Maciej Paszyński, Waldemar Rachowicz, and Adam Zdunek. *Computing with hp-Adaptive Finite Elements, Vol. II. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications*. CRC press, 2007.

[14] Jim Douglas and Henry H Rachford. "On the numerical solution of heat conduction problems in two and three space variables." In: *Transactions of the American mathematical Society* 82.2 (1956), pp. 421–439.

[15] Grazyna Hliniak Ewa Grabska. "Structural Aspects of CP-Graph Languages." In: *Schedae Informaticae* 5 (1993), pp. 81–100.

[16] Tom G. Farr et al. "The Shuttle Radar Topography Mission." In: *Reviews of Geophysics* 45.2 (2007). RG2004.

[17] Tom G. Farr et al. "The Shuttle Radar Topography Mission." In: *Reviews of Geophysics* 45.2 (2007). RG2004.

[18] Luis Ferragut, Rafael Montenegro, Gustavo Montero, Eduardo Rodríguez, M.L. Asensio, and José María Escobar. "Comparison between 2.5-D and 3-D realistic models for wind field adjustment." In: *Journal of Wind Engineering and Industrial Aerodynamics* 98.10-11 (2010), pp. 548–558. DOI: 10.1016/j.jweia.2010.04.004.

[19] Mariusz Flasiński and Robert Schaefer. "Quasi context sensitive graph grammars as a formal model of FE mesh generation." In: *Computer Assisted Mechanics and Engineering Sciences* 3.3 (1996), pp. 191–203.

[20] *GALOIS framework*. URL: http://iss.ices.utexas.edu/?p=projects/galois.

[21] Damian Goik, Konrad Jopek, Maciej Paszyński, Andrew Lenharth, Donald Nguyen, and Keshav Pingali. "Graph grammar based multi-thread multi-frontal direct solver with Galois scheduler." In: *Procedia Computer Science* 29 (2014), pp. 960–969.

[22] Nandan Gokhale, Nikos Nikiforakis, and Rupert Klein. "A dimensionally split Cartesian cut cell method for hyperbolic conservation laws." In: *Journal of Computational Physics* 364 (2018), pp. 186–208.

[23] Gene Golub and Chris Van Loan. "Matrix Computations, 3rd Edition." In: *John Hopkins University Press, Baltimore, MD* (1996).

[24] Ewa Grabska. "Theoretical Concepts of Graphical Modeling. Part Two: CP-Graph Grammars and Languages." In: *Machine Graphics and Vision* 2.2 (1993), pp. 149–178.

[25] Ewa Grabska. "Theoretical concepts of graphical modeling. Part One: Realization of CP-graphs." In: *Machine Graphics and Vision* 2.1 (1993), pp. 3–38.

[26] Ewa Grabska. "Theoretical concepts of graphical modeling. Part Two: CP-graph grammars and languages." In: *Machine Graphics and Vision* 2.2 (1993), pp. 149–178.

[27] Ewa Grabska and Grazyna Hliniak. "Structural aspects of CP-graph languages." In: *Schedae Informaticae* 5 (1993), pp. 81–100.

[28] Tara L Greaver et al. "Ecological effects of nitrogen and sulfur air pollution in the US: what do we know?" In: *Frontiers in Ecology and the Environment* 10.7 (2012), pp. 365–372. DOI: 10.1890/110049.

[29] Jean-Luc Guermond and Petar Minev. "High-order time stepping for the Navier-Stokes equations with minimal computational complexity." In: *Journal of Computational Applied Mathematics* 310 (2017), pp. 92–103.

[30] Jean-Luc Guermond, Petar Minev, and Abner J. Salgado. "Convergence analysis of a class of massively parallel direction splitting algorithms for Navier-Stokes-Boussinesq equations in simple domains." In: *Mathematics of Computation* 81.280 (2012), pp. 1951–1977.

[31] Piotr Gurgul, Marcin Sieniek, Maciej Paszyński, Łukasz Madej, and Nathan Collier. "Two-dimensional hp-adaptive Algorithm for Continuous Approximations of Material Data Using Space Projection." In: *Computer Science* 14.1 (2013), 97–112.

[32] Annegret Habel and Hans-Jörg Kreowski. "May we introduce to you: Hyperedge replacement." In: *International Workshop on Graph Grammars and Their Application to Computer Science*. Springer. 1986, pp. 15–26.

[33] Annegret Habel and Hans-Jörg Kreowski. "Some structural aspects of hypergraph languages generated by hyperedge replacement." In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 1987, pp. 207–219.

[34] Muhammad Amber Hassaan, Martin Burtscher, and Keshav Pingali. "Ordered vs. unordered: a comparison of parallelism and work-efficiency in irregular algorithms." In: *Acm Sigplan Notices* 46.8 (2011), pp. 3–12.

[35] K. He, Y. Lei, X. Pan, Y. Zhang, Q. Zhang, and D. Chen. "Co-benefits from energy policies in China." In: *Energy* 35.11 (2010), pp. 4265 –4272. DOI: 10.1016/j.energy.2008.07.021.

[36] Tobias Heuer, Peter Sanders, and Sebastian Schlag. "Network flow-based refinement for multilevel hypergraph partitioning." In: *Journal of Experimental Algorithmics (JEA)* 24 (2019), pp. 1–36.

[37] Thomas JR Hughes, Leopoldo P Franca, and Michel Mallet. "A new finite element formulation for computational fluid dynamics: VI. Convergence analysis of the generalized SUPG formulation for linear time-dependent multidimensional advective-diffusive systems." In: *Computer Methods in Applied Mechanics and Engineering* 63.1 (1987), pp. 97–112.

[38] Jim Douglas Jr. and James E. Gunn. "A general formulation of alternating direction methods. I. Parabolic and hyperbolic problems." In: *Numerical Mathematics* 6 (1964), 428–453.

[39] George Karypis. "hMETIS 1.5: A hypergraph partitioning package." In: *http://www. cs. umn. edu/~ metis* (1998).

[40] John Keating and Petar Minev. "A fast algorithm for direct simulation of particulate flows using conforming grids." In: *Journal of Computational Physics* 255 (2013), 486–501.

[41] Jozef Kelemen. "Syntactical models of cooperating/distributed problem solving." In: *Journal of Experimental and Theoretical Artificial Intelligence* 3.1 (1991), pp. 1–10.

[42] Milind Kulkarni, Keshav Pingali, Bruce Walter, Ganesh Ramanarayanan, Kavita Bala, and L Paul Chew. "Optimistic parallelism requires abstractions." In: *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2007, pp. 211–222.

[43] I. Lagzi, D. Kármán, T. Turányi, A. S. Tomlin, and L. Haszpra. "Simulation of the dispersion of nuclear contamination using an adaptive Eulerian grid model." In: *Journal of Environmental Radioactivity* 75.1 (2004), pp. 59–82. DOI: 10.1016/j.jenvrad.2003.11.003.

[44] Andrew Lenharth, Donald Nguyen, and Keshav Pingali. "Priority queues are not good concurrent priority schedulers." In: *European Conference on Parallel Processing*. Springer. 2015, pp. 209–221.

[45] Guri I Marchuk. "Splitting and alternating direction methods." In: *Handbook of numerical analysis* 1 (1990), pp. 197–462.

[46] C. Mart´n-Vide and V. Mitrana. "Cooperation in contextual grammars." In: *Proceedings of the MFCS'98 Satellite Workshop on Grammar Systems, Silesian University, Opava* (1998), pp. 289–302.

[47] Denise L. Mauzerall, Babar Sultan, Namsoug Kim, and David F. Bradford. "NOx emissions from large point sources: variability in ozone production, resulting health damages and economic costs." In: *Atmospheric Environment* 39.16 (2005), pp. 2851–2866. DOI: 10.1016/j.atmosenv.2004.12.041.

[48] Luís Monforte and Agustí Pérez-Foguet. "A multimesh adaptive scheme for air quality modeling with the finite element method." In: *Int. J. Numer. Meth. Fluids* 74.6 (2014), pp. 387–405. DOI: 10.1002/fld.3855.

[49] Luis Monforte and Agusti Pérez-Foguet. "Esquema adaptativo para problemas tridimensionales de convección-difusión." In: *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería* 30.1 (2014), pp. 60–67. DOI: 10.1016/j.rimni.2012.11.003.

[50] Rafael Montenegro, Gustavo Montero, José M. Escobar, Eduardo Rodríguez, and J. M. González-Yuste. "3-D Adaptive Wind Field Simulation Including Effects of Chimney Emissions." In: *Proceedings of WCCM VI/APCOM'04, Beijing, China*. Tsinghua University Press and Springer-Verlag, 2004.

[51] Gustavo Montero, Rafael Montenegro, and José María Escobar. "A 3-D diagnostic model for wind field adjustment." In: *Journal of Wind Engineering and Industrial Aerodynamics* 74-76.0 (1998), pp. 249–261. DOI: 10.1016/S0167-6105(98)00022-1.

[52] Gustavo Montero, Rafael Montenegro, José María Escobar, Eduardo Rodríguez, and José M. González-Yuste. "Velocity Field Modelling for Pollutant Plume Using 3-D Adaptive Finite Element Method." In: *Computational Science - ICCS 2004*. Ed. by Marian Bubak, Geert van Albada, Peter Sloot, and Jack Dongarra. Vol. 3037. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, pp. 642–645. ISBN: 978-3-540-22115-9. DOI: 10.1007/978-3-540-24687-9_92.

[53] Gustavo Montero, Eduardo Rodríguez, Rafael Montenegro, José María Escobar, and J. M. González-Yuste. "Genetic algorithms for an improved parameter estimation with local refinement of tetrahedral meshes in a wind model." In: *Advances in Engineering Software* 36.1 (2005), pp. 3–10. DOI: 10.1016/j.advengsoft.2004.03.011.

[54] Albert Oliver, Gustavo Montero, Rafael Montenegro, Eduardo Rodríguez, José María Escobar, and Agustí Pérez-Foguet. "Adaptive finite element simulation of stack pollutant emissions over complex terrains." In: *Energy* 49.0 (2013), pp. 47–60. DOI: 10.1016/j.energy.2012.10.051.

[55] Albert Oliver, Gustavo Montero, Rafael Montenegro, Eduardo Rodríguez, José María Escobar, and Agustí Pérez-Foguet. "Finite element simulation of a local scale air quality model over complex terrain." In: *Advances in Science*

and Research 8 (2012), pp. 105–113. DOI: 10.5194/asr-8-105-2012.

[56] David A Papa and Igor L Markov. "Hypergraph Partitioning and Clustering." In: *Handbook of Approximation Algorithms and Metaheuristics* 20073547 (2007), pp. 61–1.

[57] M. Pascal et al. "Assessing the public health impacts of urban air pollution in 25 European cities: Results of the Aphekom project." In: *Science of The Total Environment* 449 (2013), pp. 390–400. DOI: 10.1016/j.scitotenv.2013.01.077.

[58] Anna Paszyńska, Ewa Grabska, and Maciej Paszyński. "A Graph Grammar Model of the hp Adaptive Three dimensional Finite Element Method, Part I." In: *Fundamenta Informaticae* 114.2 (2012), pp. 149–182.

[59] Anna Paszyńska, Ewa Grabska, and Maciej Paszyński. "A Graph Grammar Model of the hp Adaptive Three dimensional Finite Element Method, Part II." In: *Fundamenta Informaticae* 114.2 (2021), pp. 183–201.

[60] Anna Paszyńska, Maciej Paszyński, and Ewa Grabska. "Graph transformations for modeling hp-adaptive Finite Element Method with triangular elements." In: *International Conference on Computational Science*. Springer. 2008, pp. 604–613.

[61] Anna Paszyńska, Maciej Paszyński, and Ewa Grabska. "Graph transformations for modeling hp-adaptive finite element method with mixed triangular and rectangular elements." In: *International Conference on Computational Science*. Springer. 2009, pp. 875–884.

[62] Maciej Paszyński and Anna Paszyńska. "Graph transformations for modeling parallel hp-adaptive Finite Element Method." In: *International Conference on Parallel Processing and Applied Mathematics*. Springer. 2007, pp. 1313–1322.

[63] Maciej Paszyński, Leszek Siwik, Krzysztof Podsiadło, and Peter Minev. "A massively parallel algorithm for the three-dimensional Navier-Stokes-Boussinesq simulations of the atmospheric phenomena." In: *Lecture Notes in Computer Science* 12137 (2020), 102–117.

[64] Jonathan A. Patz, Diarmid Campbell-Lendrum, Tracey Holloway, and Jonathan A. Foley. "Impact of regional climate change on human health." In: *Nature* 438.7066 (2005), pp. 310–317. DOI: 10.1038/nature04188.

[65] Agustí Pérez-Foguet, Albert Oliver, José María Escobar, and Eduardo Rodríguez. "Finite Element Simulation of Chimney Emissions: A Proposal for Near Field Impact Assessment in Highly Complex Terrains." In: *Procedings of the Fifth International Conference on Engineering Computational Technology*. Ed. by B.H.V. Topping, G. Montero, and R. Montenegro. paper #101. 2006. DOI: 10.4203/ccp.84.101.

[66] Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, M Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, et al. "The tao of parallelism in algorithms." In: *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*. 2011, pp. 12–25.

[67] Krzysztof Podsiadło. "https://github.com/Podsiadlo/terrain." In: (2020).

[68] V. Ramanathan and Y. Feng. "Air pollution, greenhouse gases and climate change: Global and regional perspectives." In: *Atmospheric Environment* 43.1 (2009), pp. 37–50. DOI: 10.1016/j.atmosenv.2008.09.063.

[69] Donsub Rim. "Dimensional splitting of hyperbolic partial differential equations using the Radon transform." In: *SIAM Journal on Scientific Computing* 40.6 (2018), A4184–A4207.

[70] Maria-Cecilia Rivara. "Algorithms for refining triangular grids suitable for adaptive and multigrid techniques." In: *International journal for numerical methods in Engineering* 20.4 (1984), pp. 745–756.

[71] Maria-Cecilia Rivara. "Lepp-bisection algorithms, applications and mathematical properties." In: *Applied Numerical Mathematics* 59.9 (2009), pp. 2218–2235.

[72] Maria-Cecilia Rivara. "Mesh refinement processes based on the generalized bisection of simplices." In: *SIAM Journal on Numerical Analysis* 21.3 (1984), pp. 604–613.

[73] María-Cecilia Rivara. "New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations." In: *International journal for numerical methods in Engineering* 40.18 (1997), pp. 3313–3324.

[74] Maria-Cecilia Rivara, Pedro Rodriguez, Rafael Montenegro, and Gaston Jorquera. "Multithread parallelization of lepp-bisection algorithms." In: *Applied Numerical Mathematics* 62.4 (2012), pp. 473–488.

[75] Grzegorz Rozenberg. *Handbook of graph grammars and computing by graph transformation, vol I*. Foundations. World Scientific, 1997.

[76] Iwona Ryszka, Anna Paszyńska, Ewa Grabska, Marcin Sieniek, and Maciej Paszyński. "Graph transformation systems for modeling three dimensional finite element method. Part II." In: *Fundamenta Informaticae* ().

[77] J. S. Scire, D. G. Strimaitis, and R. J. Yamartino. *A User's Guide for the CALPUFF Dispersion Model (Version 5)*. Earth Tech., Inc, Concord, MA. 2000. URL: http://www.src.com/calpuff/download/CALPUFF_UsersGuide.pdf.

[78] Marcin Sieniek and Maciej Paszyński. "Subtree reuse in multi-frontal solvers for regular grids in Step-and-Flash Imprint Nanolithography modelling." In: *Advanced Engineering Materials* (2013).

[79] Grazyna Slusarczyk and Anna Paszy ńska. "Hypergraph Grammars in hp-adaptive Finite Element Method." In: *Procedia Computer Science* 18 (2013), pp. 1545–1554.

[80] A. Spicher, O. Michel, and J. Giavitto. "Declarative Mesh Subdivision Using Topological Rewriting in MGS, International Conference on Graph Transformation, Enschede, The Netherlands." In: *Lecture Notes in Computer Science* 6372 (2010), pp. 298–313.

[81] Bruno Sportisse. "An Analysis of Operator Splitting Techniques in the Stiff Case." In: *Journal of Computational Physics* 161.1 (2000), pp. 140 –168. ISSN: 0021-9991.

[82] A.S Tomlin, S Ghorai, G Hart, and M Berzins. "3-D Multiscale air pollution modelling using adaptive unstructured meshes." In: *Environmental Modelling & Software* 15.6-7 (2000), pp. 681–692. DOI: 10.1016/S1364-8152(00)00038-4.

[83] Petr N Vabishchevich. *Additive operator-difference schemes: Splitting schemes*. Walter de Gruyter, 2013.

[84] Heather Walton, David Dajnak, Sean Beevers, Martin Williams, Paul Watkiss, and Alistair Hunt. *Understanding the Health Impacts of Air Pollution in London*. Tech. rep. TFL 90419. Environmental Research Group, School of Biomedical Sciences, King's College London, 2015. URL: http://www.kcl.ac.uk/lsm/research/divisions/aes/research/ERG/research-projects/HIAinLondonKingsReport14072015final.pdf.

[85] G. Winter, Gustavo Montero, Luis Ferragut, and Rafael Montenegro. "Adaptive strategies using standard and mixed finite elements for wind field adjustment." In: *Solar Energy* 54.1 (1995), pp. 49–56. DOI: 10.1016/0038-092X(94)00100-R.

[86] G Yarwood, S Rao, M Yocke, and G Whitten. *Updates to the Carbon Bond chemical mechanism: CB05*. Tech. rep. RT-0400675. Final report to the US EPA, 2005. URL: http://www.camx.com/publ/pdfs/cb05_final_report_120805.pdf.