Engineering with Computers (2021) 37:3857–3880
https://doi.org/10.1007/s00366-020-01253-y

ORIGINAL ARTICLE

# Parallel graph-grammar-based algorithm for the longest-edge refinement of triangular meshes and the pollution simulations in Lesser Poland area

**Krzysztof Podsiadło[1] · Albert Oliver Serra[3] · Anna Paszyńska[2] · Rafael Montenegro[3] · Ian Henriksen[4] · Maciej Paszyński[1] · Keshav Pingali[4]**

Received: 4 August 2020 / Accepted: 18 December 2020 / Published online: 22 January 2021


**Abstract**
In this paper, we propose parallel graph-grammar-based algorithm for the longest-edge refinements and the pollution simulations in Lesser Poland area. We introduce graph-grammar productions for Rivara's longest-edged algorithm for the local refinement of unstructured triangular meshes. We utilize the hyper-graph to represent the computational mesh and the graph-grammar productions to express the longest-edge mesh refinement algorithm. The parallelism in the original Rivara's longest edge refinement algorithm is obtained by processing different longest edge refinement paths in different three ads. Our graph-grammar-based algorithm allows for additional parallelization within a single longest-edge refinement path. The graph-grammar-based algorithm automatically guarantees the validity and conformity of the generated mesh; it prevents the generation of duplicated nodes and edges, elongated elements with Jacobians converging to zero, and removes all the hanging nodes automatically from the mesh. We test the algorithm on generating a surface mesh based on a topographic data of Lesser Poland area. The graph-grammar productions also generate the layers of prismatic three-dimensional elements on top of the triangular mesh, and they break each prismatic element into three tetrahedral elements. Next, we propose graph-grammar productions generating element matrices and right-hand-side vectors for each tetrahedral element. We utilize the Streamline Upwind Petrov–Galerkin (SUPG) stabilization for the pollution propagation simulations in Lesser Poland area. We use the advection–diffusion-reaction model, the Crank–Nicolson time integration scheme, and the graph-grammar-based interface to the GMRES solver.

**Keywords** Unstructured grids · Longest edge refinement · Graph-grammar · Pollution simulations · Advection–diffusion equation

## 1 Introduction

Air pollution is receiving a lot of interest nowadays. It is visible, especially in Lesser Poland area, as this is one of the most polluted cities in Europe [1]. Air pollution depends on traffic, climate, heating of building in the winter, the city's architecture, etc. The air quality can vary significantly over a distance of even a few hundred meters. Air quality simulation is a multidisciplinary endeavor. It applies numerical methods for simulations of different meteorological and chemical models [25]. This paper proposes a parallel graph-grammar-based system for simulation and prediction of air pollution over prescribed terrain data.

Most computer-aided simulations start with mesh generation of the domain with a finite set of elements. For irregular geometries, the triangular elements in two dimensions or

✉ Maciej Paszyński
maciej.paszynski@agh.edu.pl

[1] AGH University of Sciences and Technology Faculty of Computer Science, Electronics and Telecommunication Institute of Computer Science, al. A Mickiewicza 30, 30-059 Kraków, Poland

[2] Jagiellonian University Faculty of Physics, Astronomy and Applied Computer Science, Kraków, Poland

[3] University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain

[4] The University of Texas, Oden Institute, Austin, USA

tetrahedral elements in three dimensions are probably the most used finite elements in engineering computations [3]. The construction of the computational mesh usually starts from an initial mesh. It refines the mesh iteratively towards a final mesh, where we can solve our engineering problem with the required accuracy.

The process of refinement can generate so-called hanging nodes [4, 5]. In two-dimensions, they represent an edge with one triangular element subdivided, and on the other side, the large unbroken element. These nodes are difficult to handle since we have shape functions spread over the finite elements. In the hanging node case, we have to deal with the matching of approximation of "small" shape functions spread over the two broken elements with the approximation of "large" shape functions spread over the large unbroken element. In three-dimensions, the situation gets even more complicated since we may have an edge adjacent to two "small" broken tetrahedra and adjacent to many (even hundreds) of unbroken "big" elements.

We can eliminate a hanging node of a triangular element by breaking the element and connecting the hanging node with the opposite node. We must perform this algorithm in a smart way since we do not want to end up with elongated elements, where the Jacobians go to zero. One interesting example of such an algorithm, which is considered a reference for two-dimensional grids, is the Rivara longest-edge refinement algorithm [20, 31, 32].

In this paper, we express the two-dimensional Rivara algorithm using graph-grammar productions, we extend it to model the generation of the three-dimensional tetrahedral meshes. We also propose graph-grammar productions expressing the stabilized finite element method for the non-stationary advection–diffusion-reaction simulations. We incorporate the Crank–Nicolson time integration scheme and interface with GMRES solver. We utilize this parallel graph-grammar-based system for the pollution simulations in Lesser Poland area.

The authors [7] proposed the first attempt to model mesh transformations by applying the graph-grammar concept for the regular triangular two-dimensional meshes with the $h$ adaptation. The authors used quasi-context sensitive graph grammar. This approach, however, generated hanging nodes, with all the difficulties related to managing these nodes.

Another attempt utilized the Composite Programmable graph grammars (CP-graph grammar) introduced originally by [11–13] as a tool for a formal description of various design processes. The authors [10, 27–29] applied the CP-graph grammars to model two- and three-dimensional adaptive grids with hanging nodes.

In this paper, we use the concept of a hypergraph, defined in Sect. 2. The hypergraphs and their grammars have been initially introduced by [15, 16] for applications in computer graphics. There are special algorithms developed and optimized for the hypergraphs [19, 22, 26]. This paper utilizes the hypergraphs for modeling mesh refinement algorithm and interfacing with the GMRES iterative solver. We have implemented our graph-grammar-based system in GALOIS [2, 8, 14, 24, 30] framework, allowing for concurrent graph processing.

We use the topographical database Shuttle Radar Topography Mission [6] to generate in an adaptive way the two-dimensional triangular mesh representing Lesser Poland area. We utilize the longest-edge refinements, and we remove the hanging nodes. We extend the topographic mesh to a three-dimensional tetrahedral mesh, representing the air above the terrain.

The resulting three-dimensional mesh is subject to the computer simulation with the advection–diffusion-reaction time-dependent solver modeling the air pollution propagation. We employ the Streamline Upwind Petrov–Galerkin stabilization [21] of the advection–diffusion-reaction problem. We use the graph-grammar productions, generating element matrices and right-hand-side vectors for each tetrahedral element. We incorporate the Crank–Nicolson time integration scheme and interface with GMRES iterative solver.

The motivations for developing the graph-grammar-based simulation system are the following. We will compare the computational costs of the classical longest-edge refinement algorithm [20], to our graph-grammar-based algorithm on a model example. We will count the number of basic operations, such as checking the status of a single triangle and breaking a single triangle. While it is impossible to derive a formula for the computational cost for a general mesh, we will compare the algorithms on a representative model example. Classical Rivara's algorithm allows for parallelization by assigning each longest edge path to a single core. In our graph-grammar-based algorithm, we move forward, and we allow for processing a single longest edge path with multiple cores.

We also claim that developing a graph-grammar-based system has some potential benefit for parallelization of the computations. The graph grammar productions are basic undividable tasks that can be executed concurrently. The graph-grammar model allows for implementation in the graph processing system like, e.g., GALOIS environment [30]. The parallelization is obtained for free since the GALOIS system automatically manages concurrent processing of graph-grammar productions. In this paper, we implemented a graph-grammar-based model of mesh generation and generation of elemental matrices. In future work, we plan to develop the graph-grammar model of the iterative solver. We will perform matrix-vector multiplications element wise, multiplying the elemental matrices by local portions of the right-hand side, without assembling the matrices but assembling the resulting vector.

The structure of the paper is the following. We start from the formal introduction of the hypergraph and graph grammar in Sect. 2. Next, in Sect. 3, the two-dimensional mesh refinement algorithm, initially proposed by Rivara, is described. In Sect. 4, we introduce the graph grammar model expressing the Rivara mesh refinement algorithm. Section 5 is devoted to comparing the graph-grammar-based algorithm with Rivara's longest edge refinement algorithm, including the discussion on parallelization. Section 6 presents the graph-grammar-based productions interfacing with the GMRES solver, using the Crank–Nicolson time integration scheme. Finally, Sect. 7 is devoted to the numerical results of the simulation of pollution in Lesser Poland area. We conclude the paper with Sect. 8.

## 2 Hypergraphs and graph grammar

In this section, the concept of hypergraph and graph grammar are summarized, which are later used to model the refinements of the two-dimensional unstructured mesh.

**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 $E_2 \subseteq E_1$.

Figure 1 presents an exemplary hypergraph $H_1$ and its subgraph $H_2$.

**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 $(Ext = (Ext_1, Ext_2, \ldots, Ext_k))$.

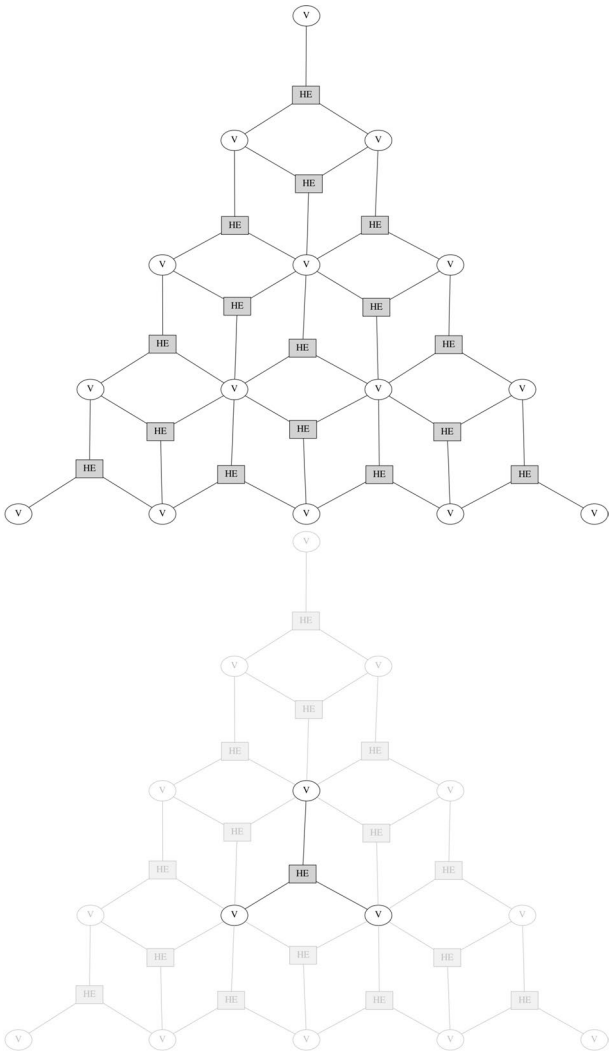


**Fig. 1** Hypergraph $G_1$ (top) and hypergraph $G_2$ (bottom). Notice that $G_2$ is a subgraph of $G_1$ (grayed out)

***Remark 1*** Let $G_2$ be a subgraph of $G_1$. 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 $H_3$. See Fig. 2.

**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 both have the same number of nodes and edges, the corresponding nodes of both graphs have the same labels and attributes, and the corresponding

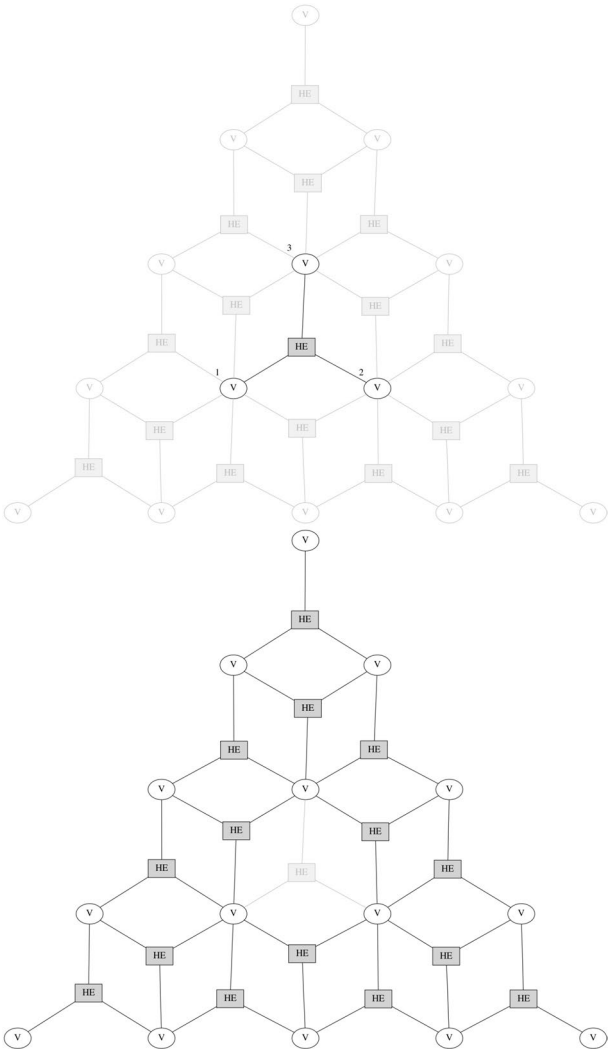

**Fig. 2** Production suitable hypergraph $H_2$ (top) and hypergraph $G_3$ (bottom). Notice that the external nodes of $H_2$ ($V_1$, $V_2$, and $V_3$) are the connections with $G_3$

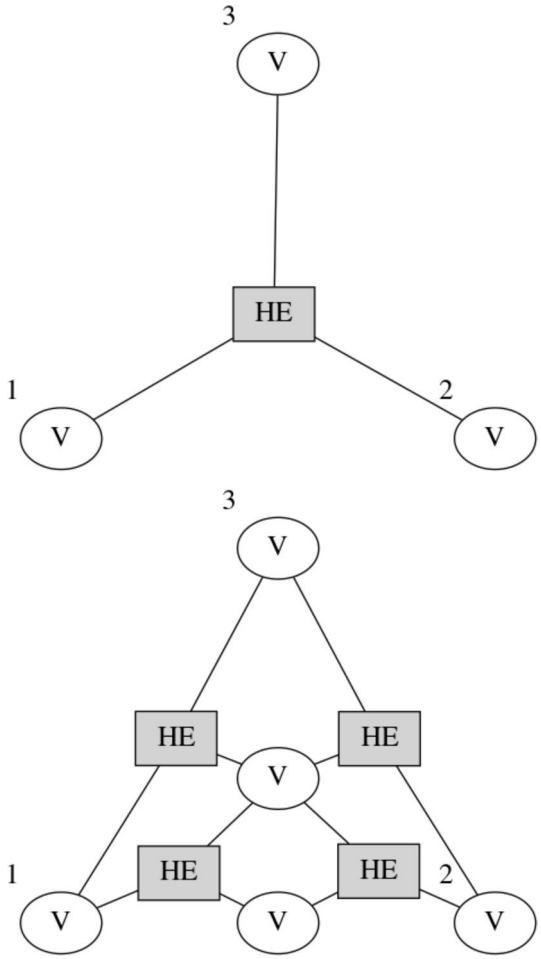


**Fig. 3** Production $p = (H_2, H_4)$ where the production suitable hypergraph $H_2$ (top) is substituted by the $H_4$ (bottom)

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

- $\big(l(v) = l'(f(v))\big)\, \forall v \in V$,
- $\big(l(e) = l'(g(e))\big)\, \forall e \in EH$ ,
- $\big(at(v) = at'(f(v))\big)\, \forall v \in V$ ,
- $\big(at(e) = at'(g(e))\big)\, \forall e \in EH$ ,
- $(v_1, v_2, \ldots, v_k) = t(e)$ 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 $EH_1 \backslash EH_2 \cup EH_4$. See Figs. 3, 4.

The definition of graph grammar production can be extended by adding a condition over the labels and values of the hypergraphs' attributes, named the applicability predicate, which determines whether a hypergraph production can be applied.

**Definition 6** A hypergraph production with applicability predicate is a triple $p = (L, R, r)$, where both $L$ and $R$ are

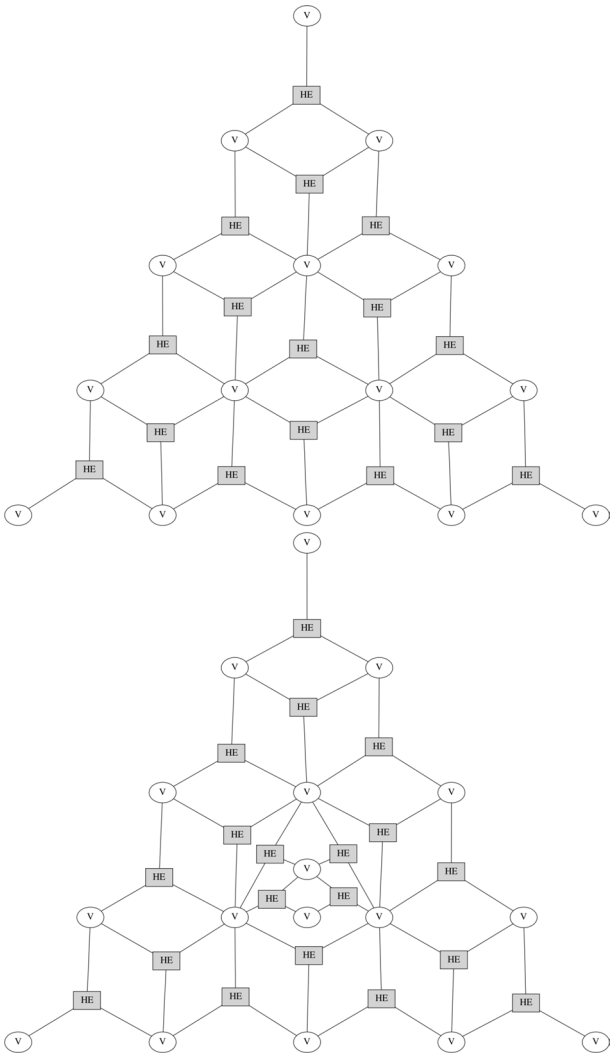


**Fig. 4** After executing the production, the original hypergraph $G_1$ (top) is converted into $G_5$ (bottom)

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.

A production with applicability predicate can be applied to a graph only in such a case in which the applicability predicate is fulfilled.

**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.

## 3 Longest-edge refinement algorithm

In this section, we are giving a brief description of the longest-edge algorithm. The longest-edge algorithm was first introduced by Rivara in 1984 [32] as the generalized 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 exist two points $a_k, a_m$ such that $\delta(q) = \delta(\langle a_k, a_m \rangle)$. Then, the generalized bisection of the simplex $q$ consists in adding a new point $a = (a_k + a_m)/2$, and splitting the simplex $q$ in two new simplices $q_k$ and $q_m$ such that

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

Once the theoretical framework of the general bisection is laid out, Rivara develops a practical algorithm to ensure the conformity of the mesh: the Longest-Edge Propagation Path (*LEPP*) [20]. The main idea is to bisect an edge only when it is the longest-edge of all its adjacent elements (this edge is known as the terminal edge). To this end, when one triangle $\tau_0$ is marked to be refined, we need to traverse all the adjacent elements through its longest edge until we find a terminal edge. All the traversal elements are known as Longest-Edge Propagation Path, and they constitute the set *LEPP* ($\tau_0$). The algorithm bisects the last two elements of the set *LEPP*($\tau_0$) and reconstructs it again until *LEPP*($\tau_0$) is empty.

In 2009, Rivara published a review of the longest-edge bisection method [17] where she describes the man properties of the method:

- The iterative and arbitrary use of this method produces triangles whose smallest interior angles are always greater than or equal to half the initial mesh's smallest internal angle. Furthermore, there exists a similarity between generated triangles. This property proves the non-degeneracy of the algorithm.
- Longest-edge bisection always terminates in a finite number of steps.
- The relationship between the two adjacent triangles' diameter is positive and greater than a constant ($K$) that depends on the initial triangulation. This property ensures the smoothness (no abrupt change of size) of the new mesh.
- The global iterative application of the method in any triangulation generates most of the new triangles quasi-equilateral (with smallest angles greater than 30°).

In 2013, a parallel multi-threaded version of the *LEPP* algorithm was developed by Rivara [18], where each thread manages the *LEPP* of a single triangle marked to be refined.

The new contributions of our paper can be summarized as follows:

- 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 Crank–Nicolson time-integration scheme and interface our graph-grammar system with GMRES solver,
- we perform the pollution simulations in Lesser Poland area.

## 4 Two-dimensional mesh refinement algorithm expressed by a hypergraph-grammar

In this section, we express the two-dimensional mesh refinement algorithm, initially proposed by Rivara [20] using 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 modify the triangles locally.

Productions are set such that all of them bisect the triangle, so they have to be applied only at triangles marked for refinement or triangles that need to be bisected to conform to the mesh; i.e., triangles with one, two, or three hanging-nodes.

***Remark 2*** Criteria for longest-edge in equilateral or isosceles triangles. The criterion for choosing the longest-edge is to prevent propagation, therefore, the priority for edges is: 1. Edge with a hanging-node; 2. Edge on the boundary; 3. Rest of the edges.

### 4.1 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 $D_R = \{TRUE, FALSE\}$.

### 4.2 Productions

We need six productions to perform the longest-edge bisection algorithm. They can be summarized as follows:

**(P1)** Triangle has no hanging-node and is marked to be refined. Predicate prioritizes the longest-edge on the border.

**(P2)** Triangle has one hanging-node; the longest-edge is the one that contains the hanging-node.

**(P3)** Triangle has one hanging-node; the longest-edge is one that does not contain the hanging-node. Predicate prioritizes the longest-edge on the border.

**(P4)** Triangle has two hanging-nodes; the longest-edge is one that contains a hanging-node.

**(P5)** Triangle has two hanging-nodes; the longest-edge is the one that does not contain a hanging-node. Predicate prioritizes the longest-edge on the border.

**(P6)** Triangle has three hanging-nodes.

***Remark 3*** The six productions assume that any triangle edge can only contain one hanging-node. To ensure this restriction, productions only allow to break one edge that is connected to two regular nodes (no hanging-nodes).

To improve readability in the productions, a new function ($NL$) has been introduced. This function computes the length of a new edge depending on nodes. There are two possibilities

depending on the number of arguments. So the two functions are:

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

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

Next, we describe in more detail the six productions and their predicates.

#### 4.2.1 Production 1 (P1)

Production **(P1)**, Fig. 5, expresses the bisection of a triangle with no hanging-nodes. This production will bisect the triangle by edge 1.

Analysis of the predicate:

(*R1 AND* ((*L1* ≥ *L2*) *AND* (*L1* ≥ *L3*))). The first condition ensures that the triangle has to be marked for refinement (*R1*), and that the edge 1 is one of the longest-edges ((*L1* ≥ *L2*) *AND* (*L1* ≥ *L3*)). If these two conditions are met, there are two cases:

- (*B1*) If edge 1 is on the boundary (*B1*), then the triangle will be bisected; prioritizing the boundary. Note that in this case, we don't have to check if they are any hanging nodes at the end of the edge. This is because there are no hanging nodes on 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 edge 1 yet. Finally, we should ensure 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, generating a new node in its midpoint ($x = (x1 + x2)/2, y = (y1 + y2)/2$, $z = (z1 + z2)/2$); this new node will be hanging if the edge is not on the boundary or a regular node if it is on the boundary (*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), and inherit the boundary flag (*B*=*B1*). Then, the triangle has to be bisected; to this end, it generates a new edge that connects the newly created node with the opposite node, computing the length of this new edge (*L*=*NL*(1,2,3)) and setting it as an interior edge (*B*=*FALSE*). Finally, it generates the new two triangles that are not marked to be refined (*R*=*FALSE*).

#### 4.2.2 Production 2 (P2)

Production **(P2)**, Fig. 6, expresses the bisection of a triangle with one hanging-nodes by the edge that contains the hanging-node.

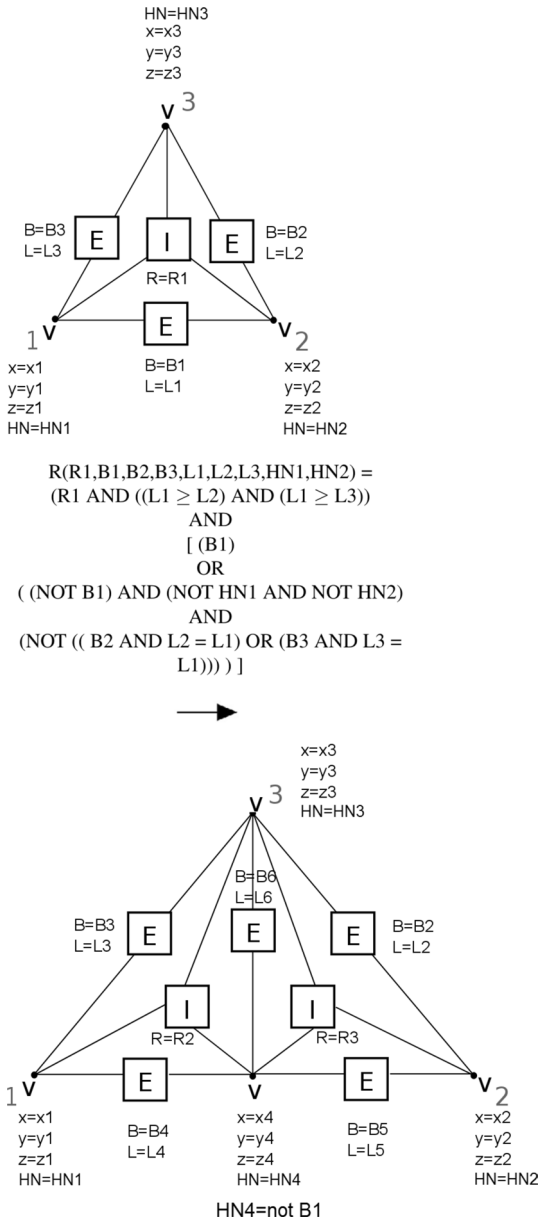


**Fig. 5** Production **(P1)** for the refinement of the marked element

Analysis of the predicate:

((*L4*+*L5*) ≥ *L2*) *AND* ((*L4*+*L5*) ≥ *L3*). The only condition in this production ensures that the broken edge (edge 4 + edge 5) is one of the longest-edges (((*L4*+*L5*) ≥ *L2*) *AND* ((*L4*+*L5*) ≥ *L3*)). 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 a higher priority.

This production does not break the longest edge since it's already broken. The production does bisect the triangle; to this end, 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 edge

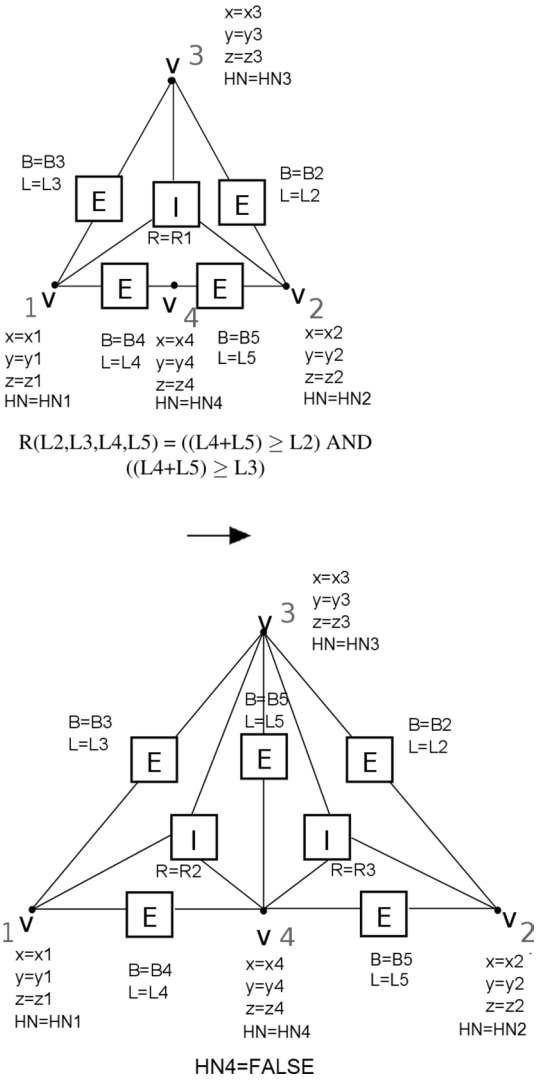


**Fig. 6** Production **(P2)** for the additional refinement of the element with the longest edge already broken with the hanging node, replacing the hanging node with the regular node

(*B=FALSE*). And finally, it generates the new two triangles that are not marked to be refined (*R=FALSE*).

#### 4.2.3 Production 3 (P3)

Production **(P3)**, Fig. 7, expresses the bisection of a triangle with one hanging-nodes by one edge that does not contain the hanging-node. This production will bisect the triangle by edge 3.

Analysis of the predicate:

( (*L3* ≥ *L2*) *AND* (*L3* > (*L4+L5*)) ). The first condition ensures that edge 3 is longer or equal to edge 2 (*L3* ≥ *L2*). It also ensures that edge 3 is strictly longer than the edge

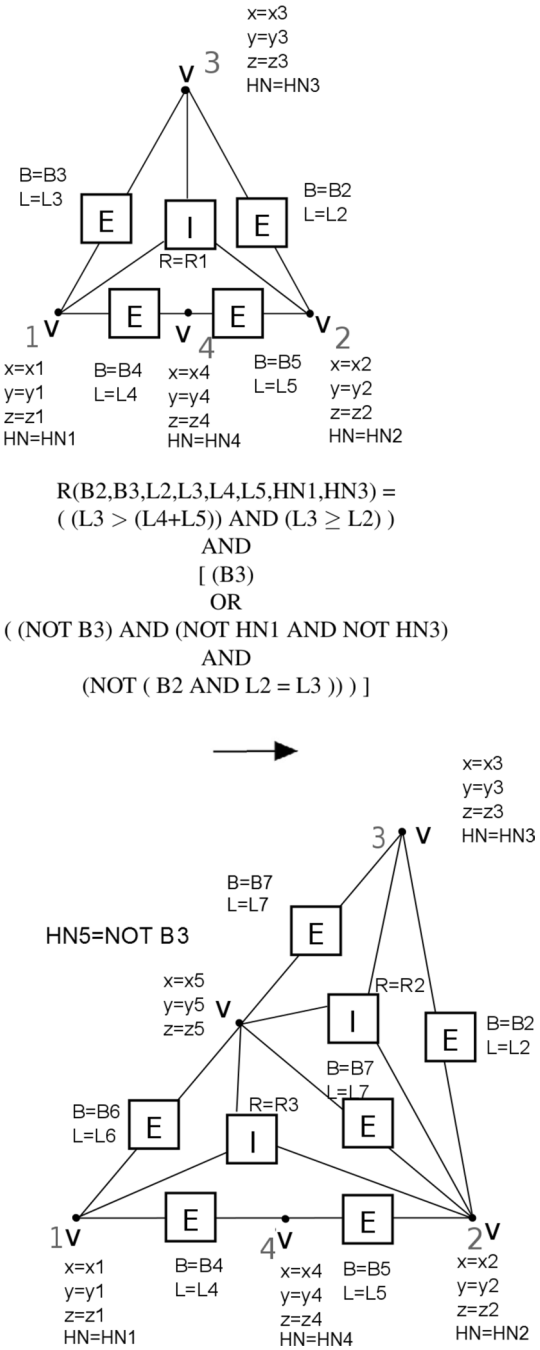


**Fig. 7** Production **(P3)** for the additional refinements of the element with one hanging node

that is broken (*L3* > (*L4+L5*)); it should be strictly longer because if they have the same length, the broken edge has higher priority.

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

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

– (*B3*) If edge 3 is on the boundary (*B3*), then the triangle will be bisected; prioritizing the boundary. Note that we don't have to check if they are any hanging nodes at the end of this edge. This is because there are no hanging nodes 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 node in its midpoint $(x = (x1 + x3)/2, y = (y1 + y3)/2, z = (z1 + z3)/2)$; this new node will be hanging if the edge is not on the boundary or a regular node if it 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), and inherit the boundary flag (*B*=*B3*). Then, the triangle has to be bisected; to this end, 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*). Finally, it generates the new two triangles that are not marked to be refined (*R*=*FALSE*).

#### 4.2.4 Production 4 (P4)

Production **(P4)**, Fig. 8, expresses the bisection of a triangle with two hanging-nodes by one of the edges that contain a hanging-node. This production will bisect the triangle by the node connected to edge 4 and edge 5.

Analysis of the predicate:

( ((*L4*+*L5*) ≥ (*L6*+*L7*)) *AND* ((*L4*+*L5*) ≥ *L3*) ). The only condition in this production ensures that the broken edge (edge 4 + edge 5) is one of the longest-edges ( ((*L4*+*L5*) ≥ (*L6*+*L7*)) *AND* ((*L4*+*L5*) ≥ *L3*) ). 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 a higher priority.

This production does not break the longest edge since it's already broken. The production does bisect the triangle; to this end, 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 edge (*B*=*FALSE*). And finally, it generates the new two triangles that are not marked to be refined (*R*=*FALSE*).

#### 4.2.5 Production 5 (P5)

Production **(P5)**, Fig. 9, expresses the bisection of a triangle with two hanging-nodes by the edge that does not contain

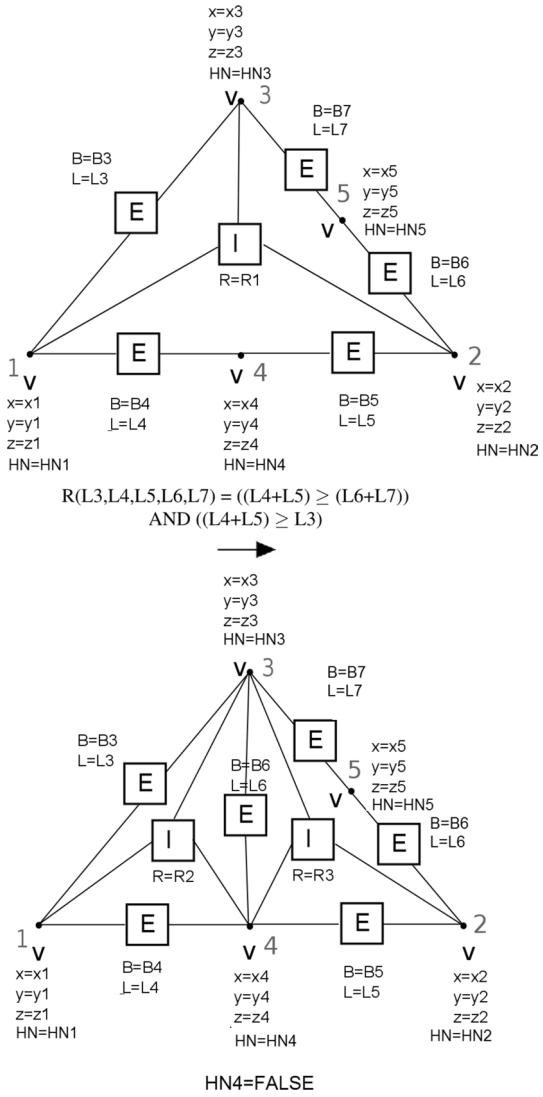


**Fig. 8** Production **(P4)** for an additional refinement of an element with two hanging nodes, breaking the element towards one of the broken edges

a hanging-node. This production will bisect the triangle by edge 3.

Analysis of the predicate:

( ( (*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 strictly longer than the other two broken edges ( (*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*); if they are, we cannot break edge 3 yet. 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, generating a new node in its midpoint$(x = (x1 + x3)/2, y = (y1 + y3)/2, z = (z1 + z3)/2)$;

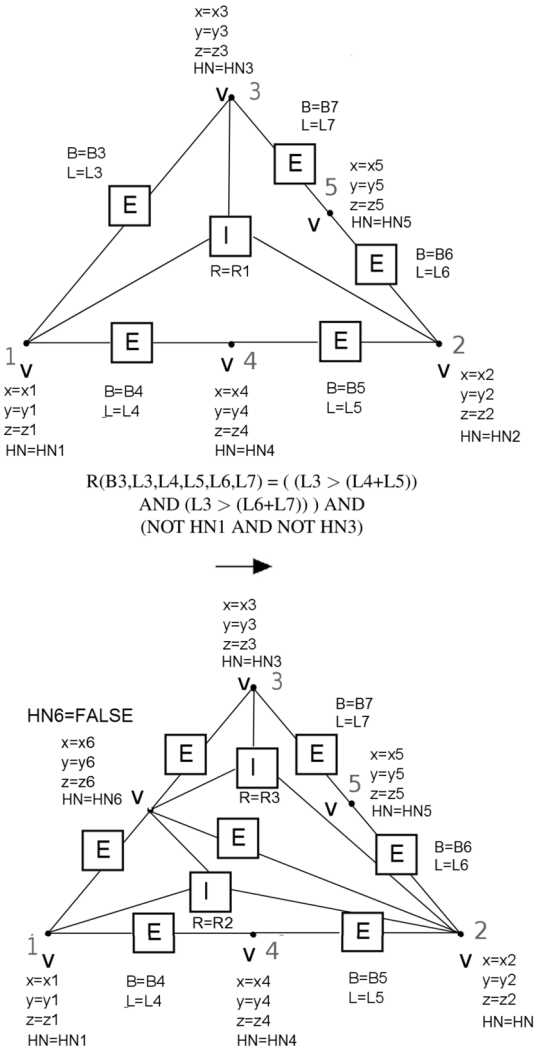


**Fig. 9** Production **(P5)** for an additional refinement of the element with two hanging nodes, breaking the element towards the unbroken edge

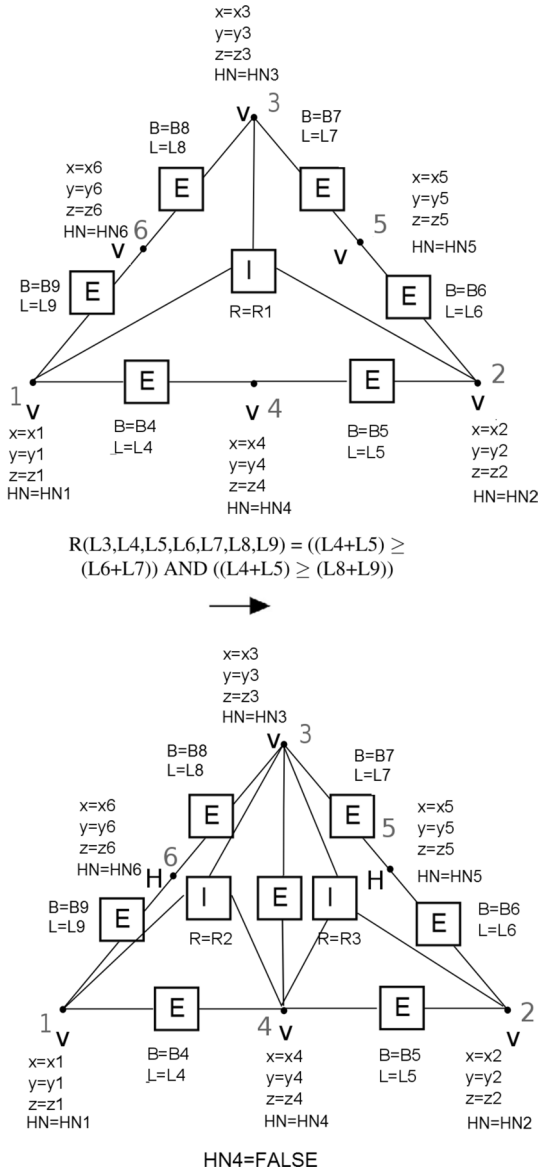


**Fig. 10** Production **(P6)** removing the hanging node from the longest edge of the element with three hanging nodes

this new node will be hanging if the edge is not on the boundary, or a regular node if it 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), and inherit the boundary flag (*B*=*B3*). Then, the triangle has to be bisected; to this end, 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*). Finally, it generates the new two triangles that are not marked to be refined (*R*=*FALSE*).

#### 4.2.6 Production 6 (P6)

Production **(P6)**, Fig. 10, expresses the bisection of a triangle with three hanging-nodes. This production will bisect the triangle by the node connected to edge 4 and edge 5.

Analysis of the predicate:

( ((*L4*+*L5*) ≥ (*L6*+*L7*)) *AND* ((*L4*+*L5*) ≥ (*L8*+*L9*)) ). The only condition in this production ensures that the broken edge (edge 4 + edge 5) is one of the longest-edges ( ((*L4* +*L5*) ≥ (*L6*+*L7*)) *AND* ((*L4*+*L5*) ≥ (*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, since it's already broken. The production does bisect the triangle; to this end, 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 edge (*B*=*FALSE*). And finally, it generates the new two triangles that are not marked to be refined (*R*=*FALSE*).

## 5 Comparison of the longest-edge refinement algorithm and graph-grammar-based refinement algorithm

In this section, we will compare two algorithms, the classical Rivara's lonest-edge refinement algorithm, and our graph-grammar-based refinement algorithm.

To describe the Rivara algorithm, we recall the definition of $LEPP(t)$, the definition of a pair of terminal triangles, and the definition of a terminal boundary triangle.

For any triangle $t_0$ of any conforming triangulation $T$, the $LEPP(t_0)$ consists of the ordered list of all triangles $t_0, t_1, t_2, ..., t_{n-1}$, such that triangle $t_i$ is the neighbour triangle of $t_{i-1}$ by the longest edge of $t_{i-1}$, for $i = 1, 2, ..., n$.

A pair of terminal triangles are two adjacent triangles $(t, t^*)$ with a common longest edge.

A terminal 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.

**Algorithm 1** Backward-Longest-Edge-Bisection

**Require:** *t triangle to refine, T mesh of triangular elements*
1: **while** $t$ remains without being bisected **do**
2: Find the$LEPP(t)$
3: $t^* =$ the last triangle of $LEPP(t)$
4: **if** $t^*$ is a terminal boundary triangle **then**
5: bisect $t^*$
6: **else**
7: bisect the last pair of terminal triangle of $LEPP(t)$
8: **end while**

We summarize in Fig. 11 the Rivara algorithm. The green triangle is the triangle denoted to break ($t_0$). The triangles belonging to the $LEPP(t_0)$ are denoted by blue color (triangles "touched" by the algorithm), the red edges are the terminal edges, the new edges created during the refinement process.

We count the number of basic operations as performed by the algorithm, defined as checking a triangle (*CHECK*) (triangles denoted by blue color), or breaking a triangle (*BREAK*) (triangles broken at the end of the *LEPP*). The number of *CHECK*s and *BREAK*s is summarized in Table 1. While the single longest edge path algorithm has no potential for parallelization, all these *CHECK*s are executed sequentially in each step. The longest-edge refinement algorithm for a single *LEPP* is sequential. Even breaking the two triangles located at the end of the path is sequential since the common edge has to be locked until the first break is finished. The parallel processing time is identical. The parallelism in the classical Rivara algorithm is obtained by processing multiple *LEPP*s in parallel, each *LEPP* in sequential [18].

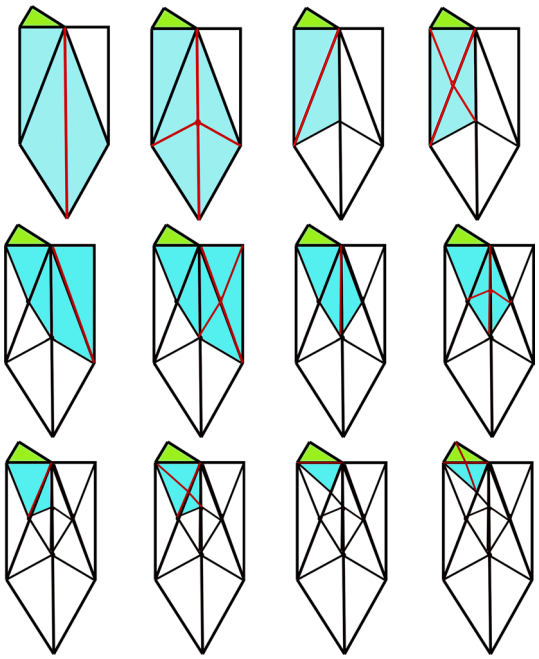

**Fig. 11** The steps of the Rivara algorithm

**Table 1** The number of touched and split triangles in each step of the Rivara algorithm presented in Fig. 11

| 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 |

The graph-grammar-based algorithm is summarized in the pseudo-code Algorithm 2. We summarize in Fig. 12 the graph-grammar-based algorithm. The initially broken triangle is denoted by green color. The checked neighbors are denoted by blue color. A red breaking line denotes the broken triangles. In Table 2 we count the number of triangles where we tried to execute the productions (*CHECK*) and the number of triangles modified by the execution of the productions (*BREAK*s). The graph-grammar algorithm has potential for parallelization even when the Rivara algorithm uses a single *LEPP*. The number of sequential *CHECK*s for graph-grammar-based algorithm is 44, but when we utilize

**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 Fig. 12

| | *CHECK*s | *BREAK*s | | |
|---|---|---|---|---|
| step | | **(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 | |

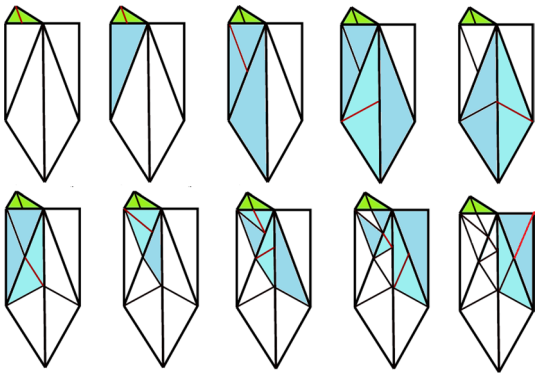

**Fig. 12** The steps of the graph-grammar-based algorithm

eight cores, it is equal to 10 steps. The number of *BREAK*s can be reduced to 9, since we break in parallel triangles that do not share the broken edge.

**Algorithm 2** Graph-grammar-based mesh refinement

**Require:** $t_0$ *triangle to refine*, *List list of triangles to refine*
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 broken **then**
8: Add to *List* neighbors of broken edges
9: continue
10: Execute production **(P3)** on triangles from $RefList$
11: **if** any triangle broken **then**
12: Add to *List* neighbors of broken edges
13: continue
14: (similarly for other productions **(P4)**, **(P5)**, **(P6)**)
15: **end while**

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

## 6 Graph-grammar-based interface with advection–diffusion-reaction solver

At the end of the two-dimensional mesh generation, we execute production **(Pmesh)** in parallel over each triangular element. It generates the three-dimensional tetrahedral elements on top of the two-dimensional triangular element.

It plots a vertical line over each of the vertices of the triangle, it partitions it into $M$ equally distances intervals, and it constructs $M$ prismatic elements, and it divides each prismatic element into three tetrahedral elements (see Fig. 13).

We construct $M$ prismatic elements, and we divide each prismatic element into three tetrahedral elements (see Fig. 13). This operation is performed by graph-grammar production (Pmesh). Next, we take a references tetrahedral element $\hat{M}$ span over $(0,0,0)-(1,0,0)$, $(0,0,0)-(0,1,0)$, $(0,0,0)-(0,0,1)$. We introduce the four basis functions

$$\hat{\psi}_1(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3) = 1-\xi_1-\xi_2-\xi_3 \tag{1}$$

$$\hat{\psi}_2(\xi_1,\xi_2,\xi_3) = \lambda_2(\xi_1,\xi_2,\xi_3) = \xi_1 \tag{2}$$

$$\hat{\psi}_3(\xi_1,\xi_2,\xi_3) = \lambda_3(\xi_1,\xi_2,\xi_3) = \xi_2 \tag{3}$$

$$\hat{\psi}_4(\xi_1,\xi_2,\xi_3) = \lambda_4(\xi_1,\xi_2,\xi_3) = \xi_3 \tag{4}$$

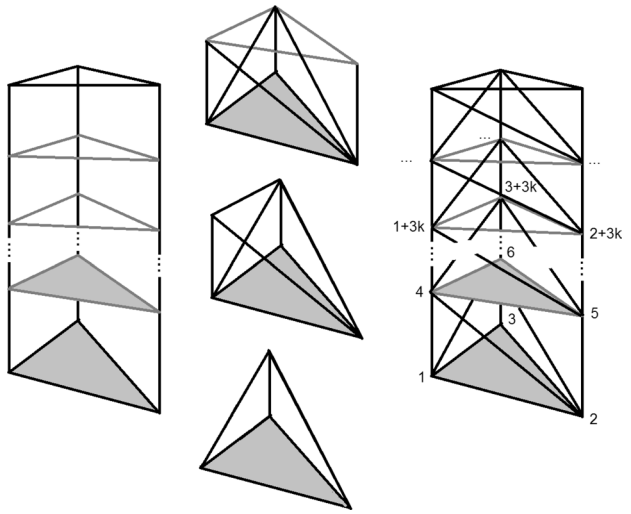


**Fig. 13** Generation of three-dimensional mesh starting from 2D mesh representing the terrain, followed by the generation of element matrices

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

$$\hat{\psi}_5(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3) \tag{5}$$

$$\hat{\psi}_6(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3) \tag{6}$$

$$\hat{\psi}_7(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3) \tag{7}$$

$$\hat{\psi}_8(\xi_1,\xi_2,\xi_3) = \lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3) \tag{8}$$

$$\hat{\psi}_9(\xi_1,\xi_2,\xi_3) = \lambda_2(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3) \tag{9}$$

$$\hat{\psi}_{10}(\xi_1,\xi_2,\xi_3) = \lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3), \tag{10}$$

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 from the reference element into an arbitrary element.

$$\hat{M} \ni (\xi_1,\xi_2,\xi_3) \rightarrow x_M(x,y,z) \in M \quad \psi_i(x,y,z) = \hat{\psi}_i(x_M^{-1}(x,y,z)) \tag{16}$$

We focus on the pollution propagation equations

$$\frac{\partial u}{\partial t} + \beta \cdot \nabla u - \nabla \cdot (\epsilon \nabla u) + cu = f \tag{17}$$

where $u(x, y, z, t)$ is the pollutant concentration field, $\beta(x,y,z,t) = \left(\beta_x(x,y,z,t), \beta_y(x,y,z,t), \beta_z(x,y,z,t)\right)$ is a given wind velocity, and $\epsilon$ is the diffusion coefficient, and $cu$ is the reaction term, see [34] for more details.

We introduce time steps $0 = t_0 < t_1 < t_2 < \cdot < t_N = T$ and we approximate the time derivative in a finite difference manner, with Crank–Nicolson scheme applied for time discretization.

$$\frac{u^{t+1} - u^t}{dt} + \beta \cdot \nabla \frac{u^{t+1} + u^t}{2} - \nabla \cdot \left(\epsilon \nabla \frac{u^{t+1} + u^t}{2}\right) + c\frac{u^{t+1} + u^t}{2} = f^t \tag{18}$$

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

$$\frac{u^{t+1} - u^t}{\Delta t} + \frac{b(u^t, v) + b(u^{t+1}, v)}{2} = l(v) \quad \forall v \in V \tag{19}$$

where

$$b(u,v) = (\beta \cdot \nabla u, v)_\Omega - (\epsilon \nabla u, \nabla v)_\Omega + (\epsilon n \cdot \nabla u, v)_\Gamma + (cu, v)_\Omega \tag{20}$$

$$l(v) = (f, v)_\Omega \tag{21}$$

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

We introduce the finite element discretization. We seek for $u_h \in V_h \subset V$

$$\left(\frac{u_h^{t+1} - u_h^t}{\Delta t}, v_h\right) + \frac{b(u_h^t, v_h) + b(u_h^{t+1}, v_h)}{2} = l(v_h) \quad \forall v_h \in V_h \subset V \tag{22}$$

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

A commonly used stabilization technique is the Streamline-upwind Petrov–Galerkin (SUPG) method [21]. In this method, we modify the weak form as follows

$$b(u_h^{t+1}, v_h) + \sum_K (R(u_h^{t+1}), \tau\beta \cdot \nabla v_h)_K = l(v_h) + \sum_K (f, \tau\beta \cdot \nabla v_h)_K \quad \forall v \in V \tag{23}$$

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 $\epsilon$ stands for the diffusion term, and $\beta = (\beta_x, \beta_y, \beta_z)$ for the convection term, and $h_K^x, h_K^y$ and $h_K^z$ are three dimensions of an element $K$. Thus, we have

$$\begin{aligned}
&b_{SUPG}(u_h^{t+1}, v_h) = l_{SUPG}(v_h) \quad \forall v_h \in V_h \\
&b_{SUPG}(u_h^{t+1}, v_h) = \beta_x\left(\frac{\partial u_h^{t+1}}{\partial x}, v_h\right)_\Omega + \\
&\beta_y\left(\frac{\partial u_h^{t+1}}{\partial y}, v_h\right)_\Omega + \beta_z\left(\frac{\partial u_h^{t+1}}{\partial z}, v_h\right)_\Omega + \\
&\epsilon\left(\frac{\partial u_h^{t+1}}{\partial x}, \frac{\partial v_h}{\partial x}\right)_\Omega + \epsilon\left(\frac{\partial u_h^{t+1}}{\partial y}, \frac{\partial v_h}{\partial y}\right)_\Omega + \\
&\epsilon\left(\frac{\partial u_h^{t+1}}{\partial z}, \frac{\partial v_h}{\partial z}\right)_\Omega + (cu_h, v_h)_\Omega - \left(\epsilon\frac{\partial u_h^{t+1}}{\partial x}n_x, v_h\right)_\Gamma - \\
&\left(\epsilon\frac{\partial u_h^{t+1}}{\partial y}n_y, v_h\right)_\Gamma - \left(\epsilon\frac{\partial u_h^{t+1}}{\partial z}n_z, v_h\right)_\Gamma + \\
&\left(\beta_x\frac{\partial u_h^{t+1}}{\partial x} + \beta_y\frac{\partial u_h^{t+1}}{\partial y} + \beta_z\frac{\partial u_h^{t+1}}{\partial z} + \epsilon\Delta u_h^{t+1},\right. \\
&\left.+\left(\frac{1}{h_x} + 3\epsilon\frac{p^2}{h_K^{x\,2} + h_K^{y\,2}}\right)^{-1}\beta_x\frac{\partial v_h}{\partial x} + \beta_y\frac{\partial v_h}{\partial y} + \beta_z\frac{\partial v_h}{\partial z}\right)_\Omega \\
&l_{SUPG}(v_h) = (f, v_h)_\Omega \\
&+\left(f, \left(\frac{1}{h_x} + 3\epsilon\frac{p^2}{h_K^{x\,2} + h_K^{y\,2}}\right)^{-1}\right. \\
&\left.\left(\beta_x\frac{\partial v_h}{\partial x} + \beta_y\frac{\partial v_h}{\partial y} + \beta_z\frac{\partial v_h}{\partial z}\right)\right)_\Omega.
\end{aligned} \tag{24}$$

Finally, we iterate with time steps with implicite Crank–Nicolson method

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

We introduce a graph grammar productions **(PgenSUPG)**, **(PgenRHS)**, **(PgenMass)** that generate the element matrix, the right-hand-side, and the mass matrix with the solution from the previous time step

$$(PgenSUPG) \quad \begin{bmatrix} b_{SUPG}^K(\psi_1, \psi_1) & \cdots & b_{SUPG}^K(\psi_1, \psi_{15}) \\ \vdots & \vdots & \vdots \\ b_{SUPG}^K(\psi_{15}, \hat{\psi}_1) & \cdots & b_{SUPG}^K(\psi_{15}, \psi_{15}) \end{bmatrix}$$

$$(PgenRHS) \quad \begin{bmatrix} l_{SUPG}^K(\psi_1) \\ \vdots \\ l_{SUPG}^K(\psi_{15}) \end{bmatrix} \tag{25}$$

$$(PgenMass) \quad \begin{bmatrix} (\psi_1, \psi_1) & \cdots & (\psi_1, \psi_{15}) \\ \vdots & \vdots & \vdots \\ (\psi_{15}, \psi_1) & \cdots & (\psi_{15}, \psi_{15}) \end{bmatrix} \tag{26}$$

These productions are executed in parallel over each element at the beginning of each time step. The results of these productions are some matrices and vectors, and they are used to construct the local system over each element, with matrices

$$\begin{bmatrix} (\psi_1, \psi_1) & \cdots & (\psi_1, \psi_{15}) \\ \vdots & \vdots & \vdots \\ (\psi_{15}, \psi_1) & \cdots & (\psi_{15}, \psi_{15}) \end{bmatrix} + \frac{\Delta t}{2} * \begin{bmatrix} b_{SUPG}^K(\psi_1, \psi_1) & \cdots & b_{SUPG}^K(\psi_1, \psi_{15}) \\ \vdots & \vdots & \vdots \\ b_{SUPG}^K(\psi_{15}, \psi_1) & \cdots & b_{SUPG}^K(\psi_{15}, \psi_{15}) \end{bmatrix} \tag{27}$$

and right-hand-sides

$$\begin{bmatrix} (\psi_1, \psi_1) & \cdots & (\psi_1, \psi_{15}) \\ \vdots & \vdots & \vdots \\ (\psi_{15}, \psi_1) & \cdots & (\psi_{15}, \psi_{15}) \end{bmatrix} \begin{bmatrix} u_1^t \\ \vdots \\ u_{15}^t \end{bmatrix} + \frac{\Delta t}{2} * \begin{bmatrix} b_{SUPG}^K(\psi_1, \psi_1) & \cdots & b_{SUPG}^K(\psi_1, \psi_{15}) \\ \vdots & \vdots & \vdots \\ b_{SUPG}^K(\psi_{15}, \psi_1) & \cdots & b_{SUPG}^K(\psi_{15}, \psi_{15}) \end{bmatrix} \begin{bmatrix} u_1^t \\ \vdots \\ u_{15}^t \end{bmatrix} + \begin{bmatrix} l_{SUPG}^K(\psi_1) \\ \vdots \\ l_{SUPG}^K(\psi_{15}) \end{bmatrix} \tag{28}$$

This is done by the production **(Psystem)** which constructs

$$\begin{aligned}
&\left((PgenSUPG) + \frac{\Delta t}{2}(PgenMass)\right)\mathbf{u}^{t+1} = \\
&(PgenMass)\mathbf{u}^t + \frac{\Delta t}{2}(PgenSUPG)\mathbf{u}^t + \\
&(PgenRHS)
\end{aligned} \tag{29}$$

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

We propose the following space refinements - time progression algorithm. We start from an initial mesh approximating the topography of the terrain roughly. We solve the first time step of the advection–diffusion-reaction problem with initial conditions. Next, we run one

iteration of the longest-edge refinement algorithm. Then, we solve the second time step of the advection–diffusion-reaction problem using the solution obtained in the previous step on the coarser mesh. We continue with the space iterations of the longest-edge refinement algorithm, and, at the same time, we iterate with time step with the advection–diffusion-reaction simulations. The general idea of the algorithm can be summarized in the pseudo-code presented in Algorithm 3. Notice that we assume some accuracy $\epsilon$ of the terrain approximation, and after reaching this accuracy, we do not perform more refinements there.

**Algorithm 3** Space refinement - time progression algorithm

**Require:** $\epsilon$, *mesh*, *initial_configuration*
1: *previous_u*:=*initial_configuration*
2: Refine *mesh* with accuracy $\epsilon$
3: **for** *time_step*=1,...,MAX_TIME_STEP **do**
4: Execute **(Pmesh)** over *mesh*
5: Execute **(PgenSUPG)** on *mesh*
6: Execute **(PgenRHS)** on *mesh*
7: Execute **(PgenMass)** on *mesh*
8: Project *previous_u* into the *mesh*
9: Execute **(Psystem)** with (27), (28) and *previous_u*
10: **call** GMRES solver to get *current_solution*
11: *previous_u*:=*current_solution*
12: **end for**

## 7 Numerical results

### 7.1 Manufactured solution advection–diffusion problem

In this section, we verify our solver by testing on the manufactured advection-dominated diffusion solver. We select the advection vector $\beta = (1, 1)^T$, and $Pe = 1/\epsilon = 100$ and solve the advection–diffusion equation with homogeneous Dirichlet boundary conditions. We utilize a manufactured solution

$$u(x, y) = \left(x + \frac{e^{Pe*x} - 1}{1 - e^{Pe}}\right)\left(y + \frac{e^{Pe*y} - 1}{1 - e^{Pe}}\right) \tag{30}$$

enforced by the forcing term $f$. We set the reaction term to zero $c = 0$. This analytic expression of the solution limits the Péclet number to $Pe = 100$ due to machine precision.

We report in Figs. 14, 15, 16 the sequence of meshes generated by the adaptive algorithm. Figure 17 presents the final mesh and the final results. We also report the convergence in $L^2$ norm in Table 3.

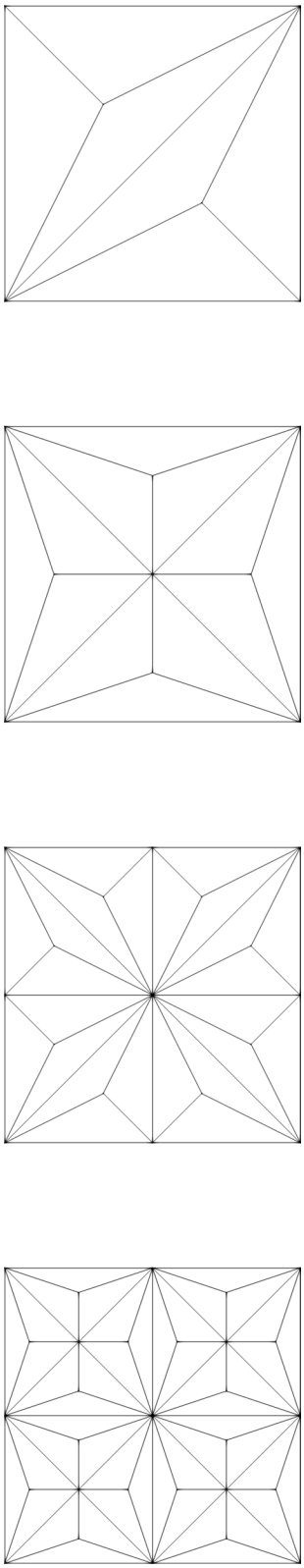

**Fig. 14** Sequence of adaptive meshes (1/3) generated for advection–diffusion manufactured solution problem stabilized with SUPG

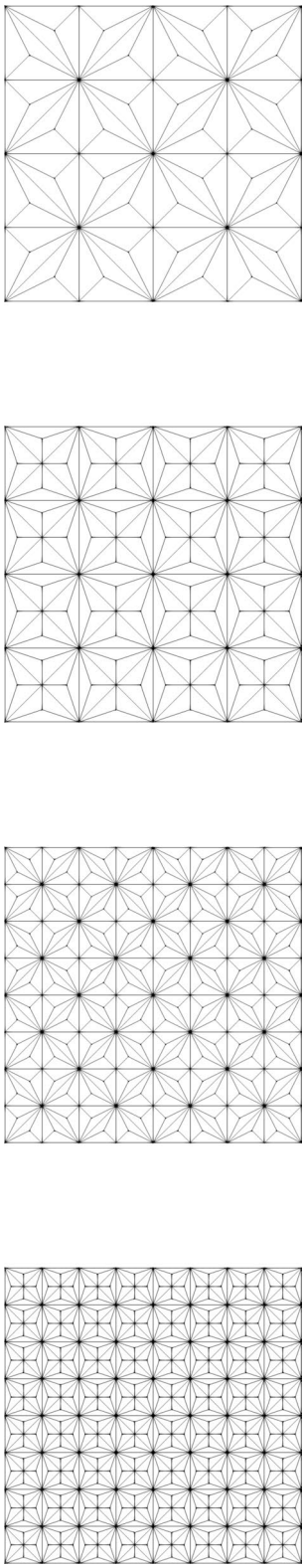

**Fig. 15** Sequence of adaptive meshes (2/3) generated for advection–diffusion manufactured solution problem stabilized with SUPG

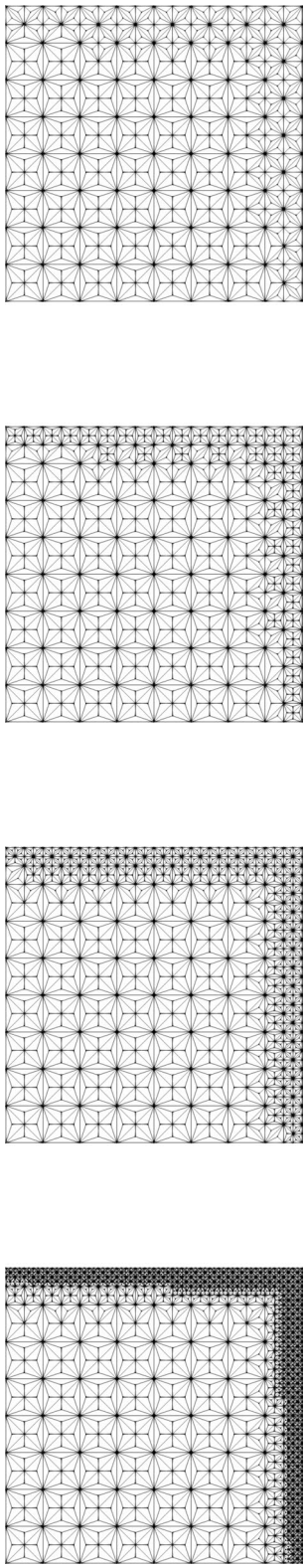

**Fig. 16** Sequence of adaptive meshes (3/3) generated for advection–diffusion manufactured solution problem stabilized with SUPG

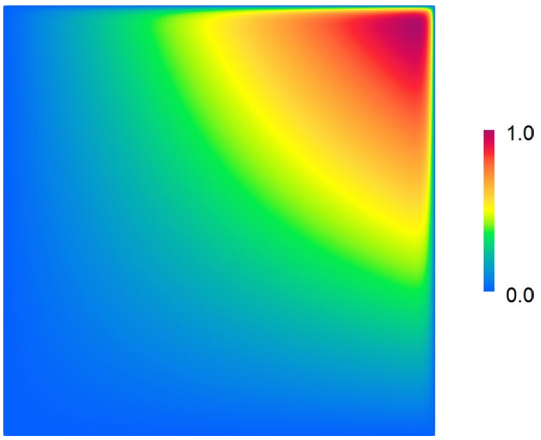


**Fig. 17** Result to the manufactured solution problem computed on the final mesh

### 7.2 Mesh generation for the pollution simulations in Lesser Poland area

We run four different experiments, starting from four different initial meshes, presented in Fig. 18. The first initial mesh is a regular triangular mesh. The second mesh is the Delaunay mesh obtained from GMSH mesh generator [9]. The third mesh is obtained from the MeshAdapt algorithm, also from the GMSH generator. The fourth one is the Frontal–Delaunay mesh from the GMSH generator. For the regular initial mesh case, the longest-edge refinement algorithm coincides with the Kossaczky refinement algorithm [23]; for other meshes, it is not equivalent to the Kosaczky algorithm.

These initial meshes are not the exact rectangles since the Earth is not flat, and the input data are taken from the Earth database [6]. The initial mesh elements have vertices adjusted to the three-dimensional points located in the $\mathcal{R}^3$ space, as read from the Earth's topography database.

We run 24 iterations of our graph-grammar based algorithm. Figures 19, 20 presents some snapshots from the refinement process. The triangulation of the terrain surface is presented in Fig. 21.

### 7.3 Computations of the wind vector field

We generate a 3D mesh on top of a 2D meshes with four layers of prisms, each divided into three tetrahedrons. We first focus on the computations of the wind distribution in the entire area, based o two fixed values of the velocity based on the real measurements from the station in Kasprowy Wierch mountain and the station in Zakopane city. We generalize these measurements into the entire domain by solving the div$u = 0$ equation. The results are presented in Fig. 22. We have the north-western wind slightly changing in time.

### 7.4 Computations of the pollution propagation

Having the advection field, representing the wind, we focus on the advection–diffusion-reaction problem.

$$\frac{\partial u_i}{\partial t} + \beta \cdot \nabla u_i - \nabla \cdot \left(K \nabla u_i\right) = s(u_i) \tag{31}$$

where $u_i(x, y, z, t)$ are the unknown concentrations of the four components of the pollution in the area, namely the vector of four unknowns, representing the chemical components $u_1 = [SO2], u_2 = [SO4], u_3 = [NO4], u_4 = [NO3], \beta(x, y, z, t)$ is a wind velocity vector computed above, representing the north-western wind, $s(u)$ is the chemical reactions part, where we assume linear model $s(u) = Au$ where $A$ is the chemical reactions matrix,

$$A = \begin{bmatrix} -0.15 & 0 & 0 & 0 \\ 0.15 & 0 & 0 & 0 \\ 0 & 0 & -0.3 & 0 \\ 0 & 0 & 0.3 & 0 \end{bmatrix} \tag{32}$$

and $K$ is the diagonal diffusion matrix

$$K = \begin{bmatrix} 8 * 10^{-6} & 0 & 0 \\ 0 & 8 * 10^{-6} & 0 \\ 0 & 0 & 4 * 10^{-6} \end{bmatrix} m^2/s \tag{33}$$

where the horizontal diffusion coefficient is equal to $8 * 10^{-6} m^2/s$, and the vertical diffusion coefficient $4 * 10^{-6} m^2/s$. The diffusion matrix is assumed to be identical for all the four species of the concentration field $u$. Since we have four components of the pollution, our basis functions, and element matrices and right-hand-side vectors are "duplicated" four times, and they are coupled now through the chemical reactions matrix. We assume that the pollutant comes from the north boundary, and it is blown inside the domain from the north boundary by the north-western wind computed based on the real measurements.

**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 |

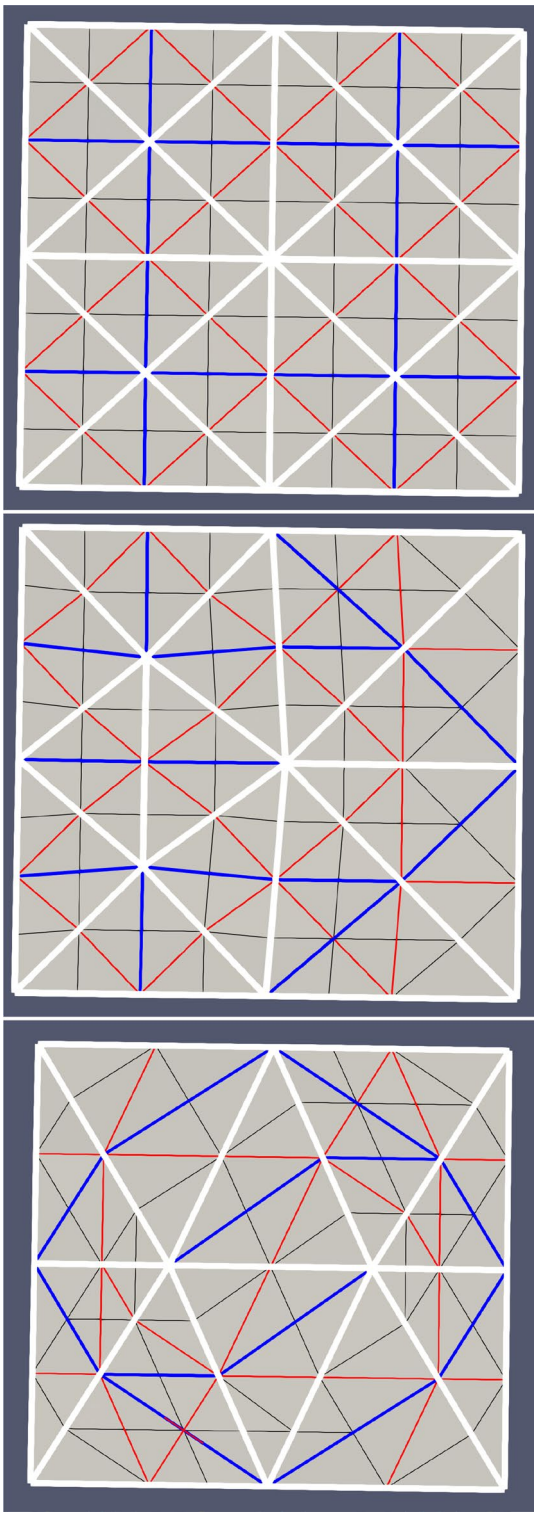

**Fig. 18** The initial coarse meshes. The first initial mesh is a regular triangular mesh, the second mesh is the Delunay mesh obtained from the GMSH mesh generator, the third mesh is obtained from the MeshAdapt algorithm, also from the GMSH generator

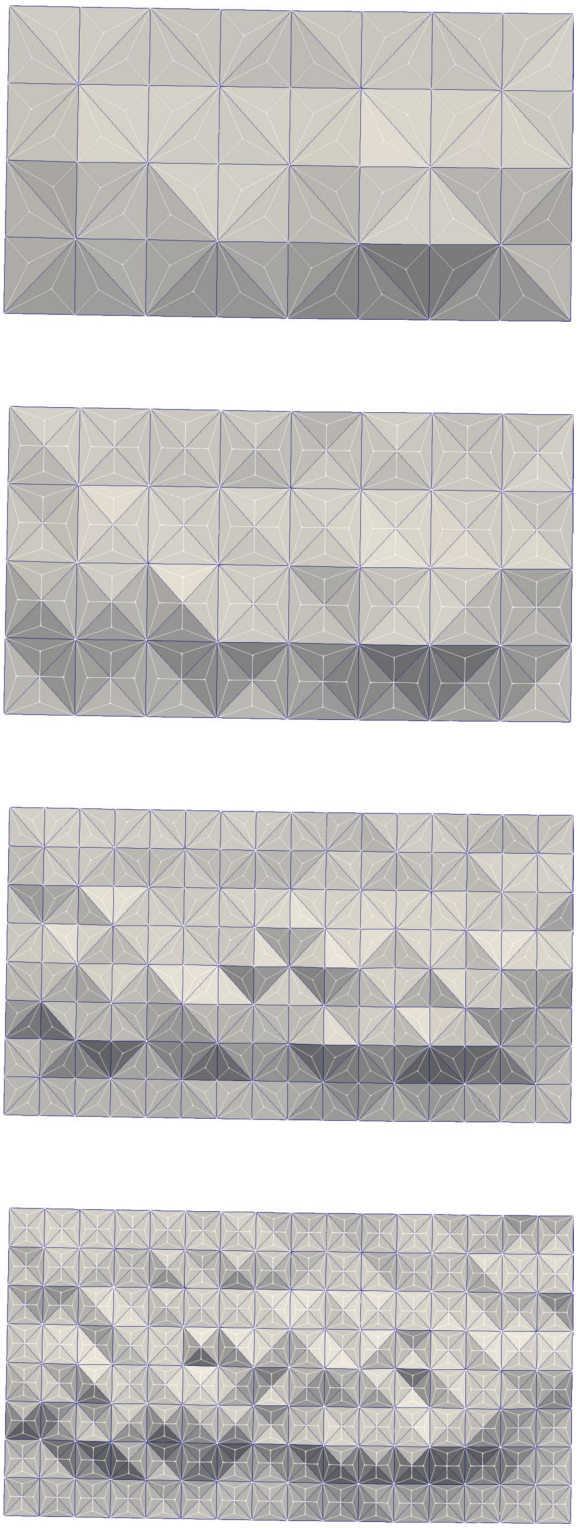

**Fig. 19** Snapshots (1/2) from the terrain 2D mesh generation

$$u_i = \beta, \tag{34}$$

and we have

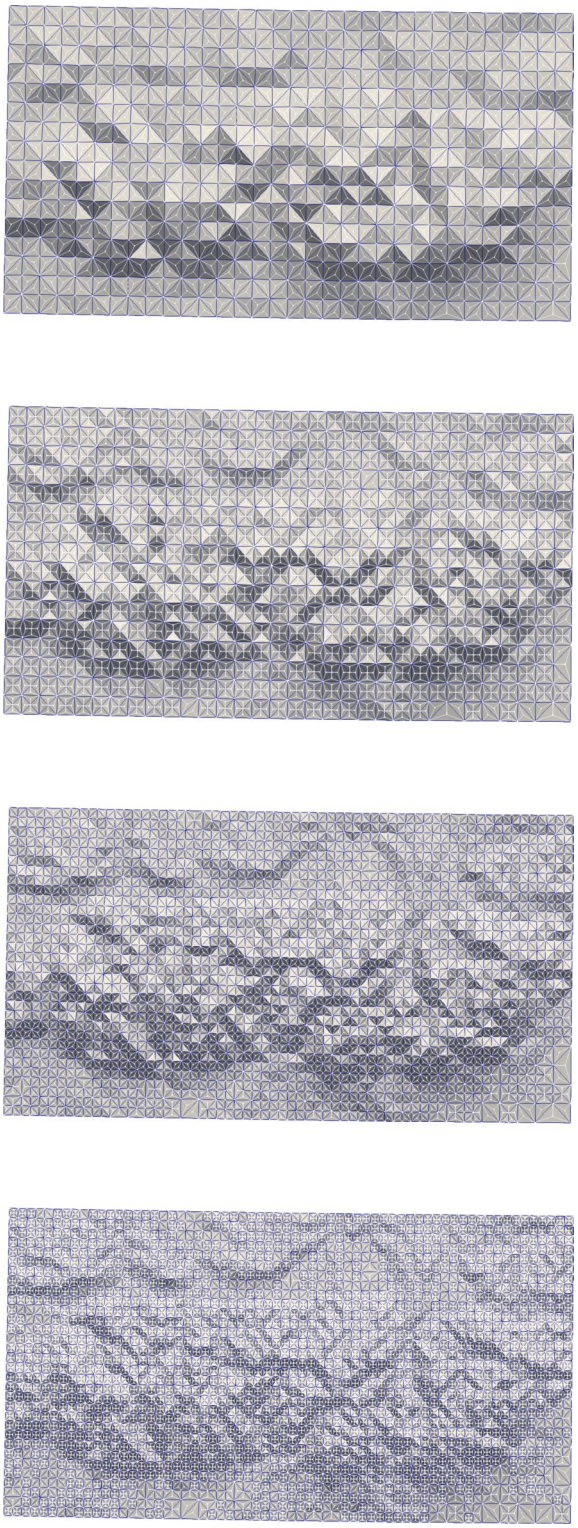

**Fig. 20** Snapshots (2/2) from the terrain 2D mesh generation

$$u(x, y, z, t) = 0, \tag{35}$$

at the wind outlet. Moreover, we have

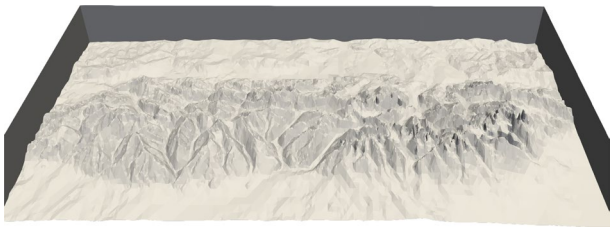

**Fig. 21** Final mesh representing Lesser District of Poland (South of Poland)

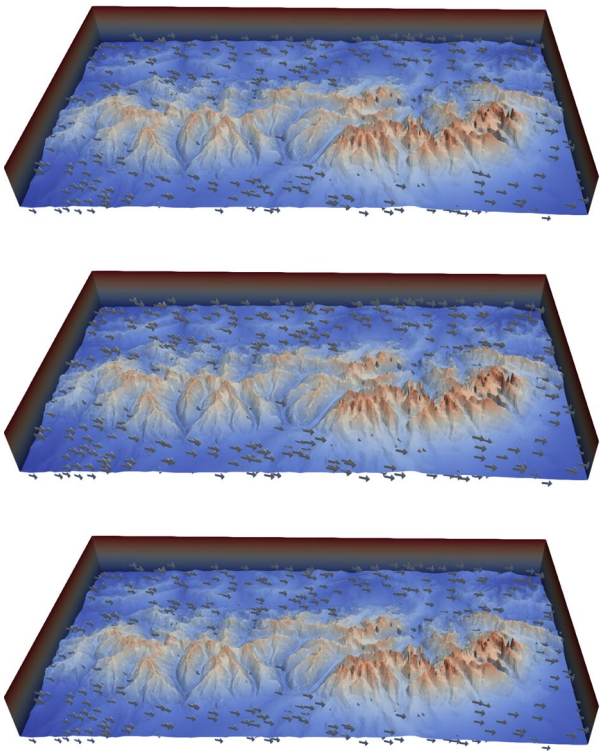

**Fig. 22** The wind vector field computed by solving div$u = 0$ based on two stations point measurements. The maximum wind speed varies between 33 and 49 km/s

$$n \cdot \left(K \nabla u_i\right) = -V^d u_i \tag{36}$$

at the terrain level, where $V^d = 1.3 * 10^{-3}$ m/s (so-called diagonal term of the deposition matrix).

We run the entire simulation on a single Linux cluster node. In Figs. 23, 24, 25, 26, we present some snapshots of the simulation. They present the propagation from the north boundary over the terrain by the north-western wind.

We have implemented our graph-grammar-based system in GALOIS [8, 30] framework, allowing for concurrent graph processing. The code is compiled on node13 on the Atari Linux cluster from the Adaptive Algorithms and Systems (a2s.agh.edu.pl) research group from the Department of Computer Science, AGH University. The

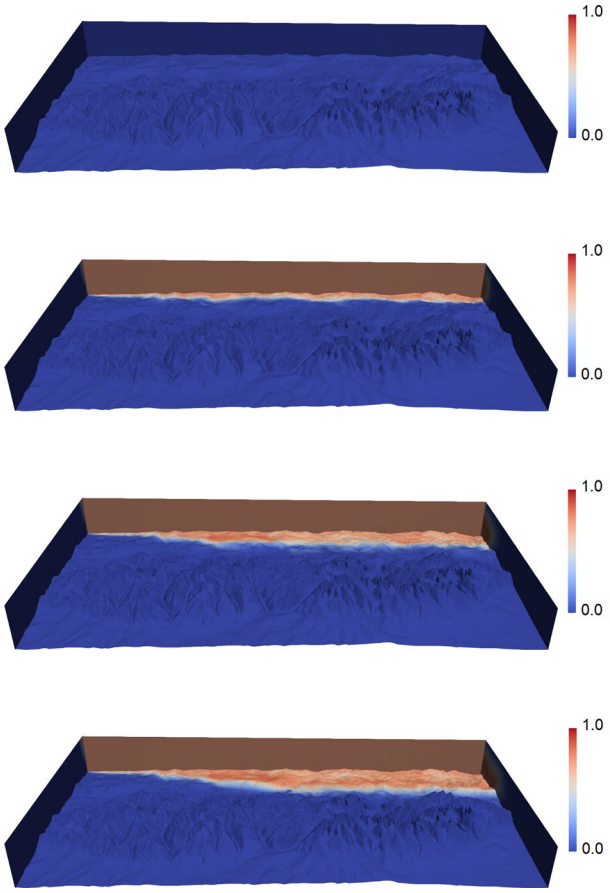


**Fig. 23** Pollution propagated by north-western wind, front view (1/2)

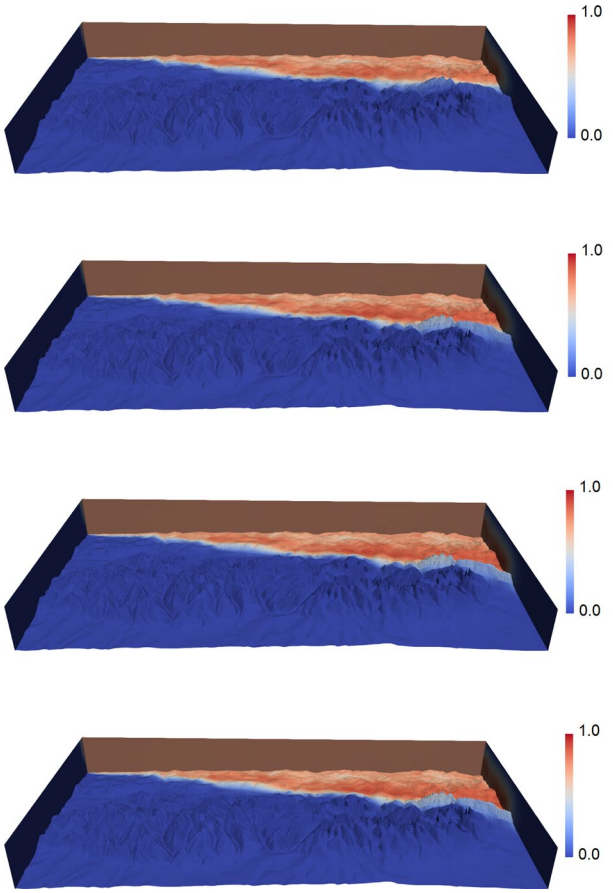


**Fig. 24** Pollution propagated by north-western wind, front view (2/2)

node13 has Intel(R) Xeon(R) CPU E5-2680 v4 processor with 2.40GHz clock and 28 cores. The code requires gcc/8.1, boost, and cmake. The code can be downloaded from *https://github.com/Podsiadlo/TerrainMesh Generator/ lonestar/graphgrammar2*. Here, we present the scalability results for the concurrent code for 24 iterations. The scalability of the code running from any of the four initial meshes is similar. We report here the timings for the regular initial mesh. It transforms the initial mesh of 24 triangles into the final mesh with over 10,000,000 triangles. In Table 4 and Fig. 27, we report the execution time on the last 15 iterations of the mesh generation algorithm, with the number of used cores increasing from 1 to 28. In Table 5 and Fig. 28, we report the speedup on the last 15 iterations of the mesh generation algorithm, with the number of used cores increasing from 1 to 28. We report the timings and speedup for iterations 15–24. The previous iterations took less than 100 milliseconds, so we neglect them from our analysis. Notice that after reaching the assumed accuracy for the terrain approximation in step 20, a single step's computational cost goes down since we do not perform massive refinements over the entire mesh.

## 8 Conclusions

This paper 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 future work will include the simulation of the pollution resulting from point sources in the Kraków city

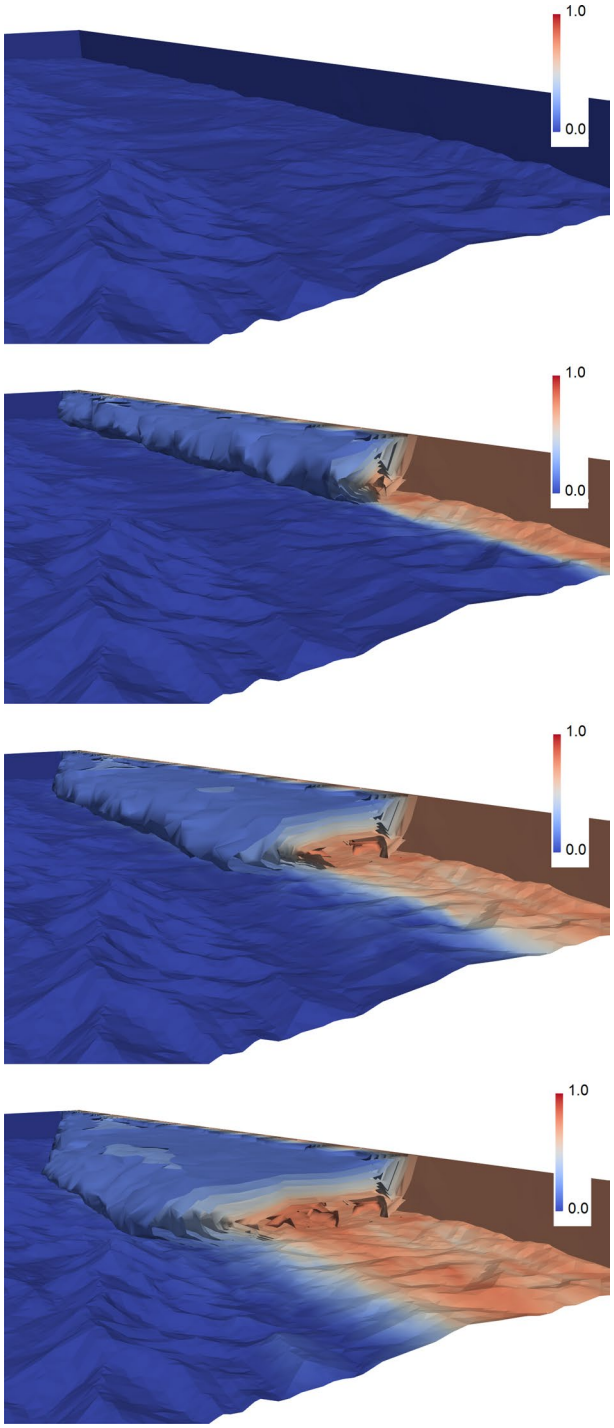


**Fig. 25** Pollution propagated by north-western wind, zoom towards right top corner, and plotting the cross-section and contours (1/2)

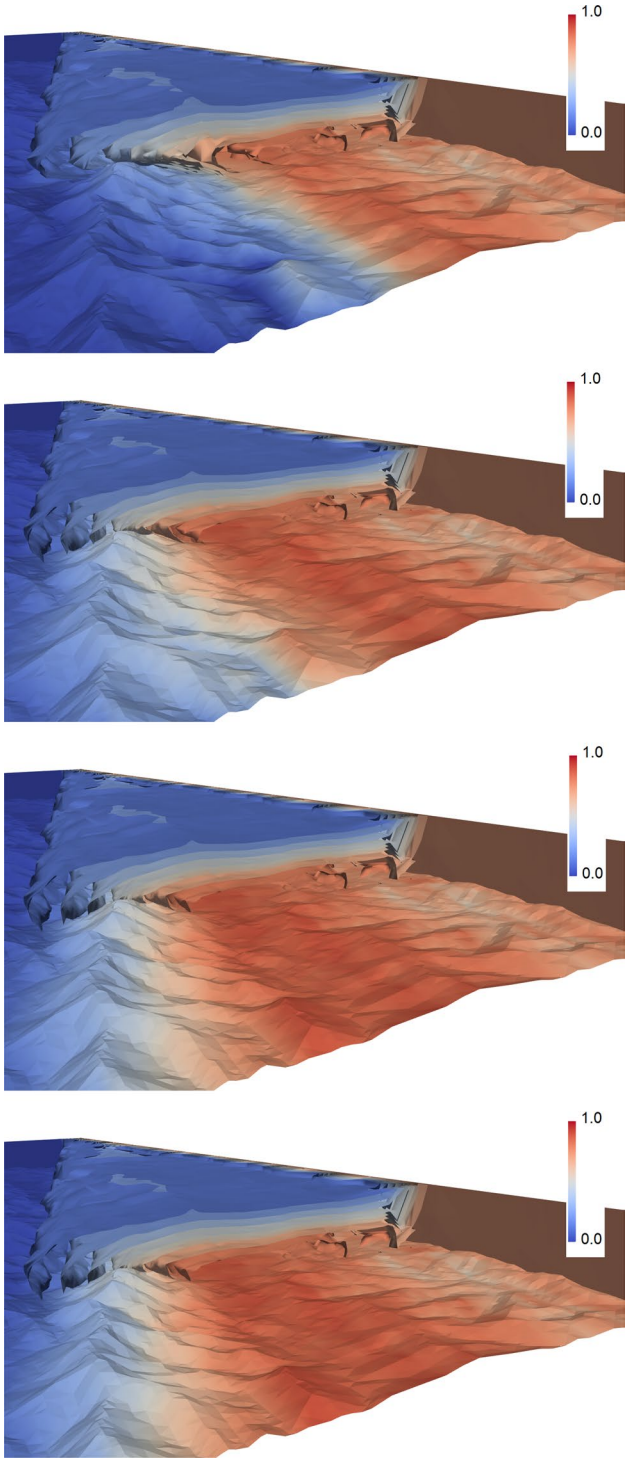


**Fig. 26** Pollution propagated by north-western wind, zoom towards right top corner, and plotting the cross-section and contours (2/2)

**Table 4** Execution times of the GALOIS mesh generator for up to 28 cores on Atari Linux cluster node, for the mesh adaptation 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 |

**Table 5** Speedup of the GALOIS mesh generator for up to 28 cores on Atari Linux cluster node, for the mesh adaptation 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 |

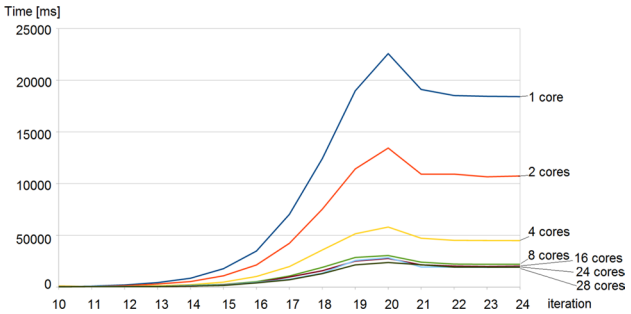


**Fig. 27** Execution times for GALOIS code on the last 15 iterations of mesh generation process

area, e.g., from the factories' chimneys. We also would like to have the thermal inversion effects simulated with Navier-Stokes-Boussinesq equations [35].

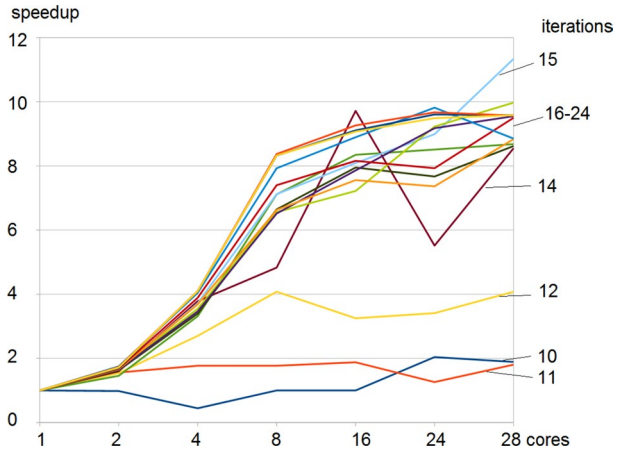


**Fig. 28** Speedup for GALOIS code on the last 3 iterations of mesh generation process

**Acknowledgements** The visit of Albert Oliver Serra at AGH have been supported by National Science Centre, Poland Grant no. 2015/17/B/ST6/01867. The work of Maciej Paszyński, Krzysztof Podsiadło has been supported by National Science Centre, Poland Grant no. 2017/26/M/ ST1/ 00281. The visit of Maciej Paszyński at Oden Institute was supported by JT Oden Research Faculty Fellowship. The work of Keshav Pingali and Ian Henriksen was supported by NSF Grants 1337281, 1406355, and 1618425, and by DARPA contracts FA8750-16-2-0004 and FA8650-15-C-7563.



## References

1. European Environment Agency, Air Quality in Europe - 2017 report
2. Andrew A, Nguyen D, Pingali K (2015) Priority queues are not good concurrent priority schedulers. European Conference on Parallel Processing, Springer, Berlin, Heidelberg 209–221
3. Carey GF, Oden JT (1984) Finite elements: computational aspects. Prentice-Hall, Upper Saddle River
4. Demkowicz L (2006) Computing with hp-Adaptive Finite Elements, vol I. Chapman & Hall / CRC Applied Mathematics & Nonlinear Science, One and Two Dimensional Elliptic and Maxwell Problems
5. Demkowicz L, Kurtz J, Pardo D, Paszyński M, Rachowicz W, Zdunek A (2007) *Computing with hp-Adaptive Finite Elements, Vol. II. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications*, Chapman & Hall / CRC Applied Mathematics & Nonlinear Science
6. Farr TG, Rosen PA, Caro E, Crippen R, Duren R, Hensley S, Kobrick M, Paller M, Rodriguez E, Roth L, Seal D, Shaffer S, Shimada J, Umland J, Werner M, Oskin M, Burbank D, Alsdorf D (2005) *The Shuttle Radar Topography Mission*, Reviews of Geophysics 45(2) *https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2005RG000183.*
7. Flasiński M, Schaefer R (1996) Quasi context sensitive graph grammars as a formal model of FE mesh generation. Comput Assist Mech Eng Sci 3:191–203
8. Galois Framework. *http://iss.ices.utexas.edu/?p=projects/galois*
9. Geuzaine C, Remacle J-F (2009) GMSH: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Meth Eng 79(11):1309–1331
10. Goik D, Jopek K, Paszyński M, Lenharth A, Nguyen D, Pingali K (2014) Graph grammar based multi-thread multi-frontal direct solver with Galois scheduler. Proc Comput Sci 29:960–969
11. Grabska E (1993a) Theoretical concepts of graphical modeling. Part one: realization of CP-graphs. Mach Graph Vis 2(1):3–38
12. Grabska E (1993) Theoretical concepts of graphical modeling. Part two: CP-graph grammars and languages. Mach Graph Vis 2(2):149–178
13. Grabska E, Hliniak G (1993) Structural aspects of CP-graph languages. Schedae Informaticae 5:81–100
14. Hassaan MA, Burtscher M, Pingali K (2011) Ordered vs. Unordered: *A comparison of parallelism and work-efficiency in irregular algorithms*, *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, PPoPP '11
15. Habel A, Kreowski HJ (1987) May we introduce to you: hyperedge replacement. Lect Notes Comput Sci 291:5–26
16. Habel A, Kreowski HJ (1987) Some structural aspects of hypergraph languages generated by hyperedge replacement. Lect Notes Comput Sci 247:207–219
17. Rivara MC (2009) Lepp-bisection algorithms, applications and mathematical properties. Appl Numer Math 59(9):2218–2235
18. Rivara MC, Rodriguez P, Montenegro R, Jorquera G (2012) Multithread parallelization of Lepp-bisection algorithms. Appl Numer Math 62(4):473–488
19. Heuer T, Sanders PG, Schlag S (2019) Network flow-based refinement for multilevel hypergraph partitioning. J Exp Algorithmics 24(1):1–36
20. Rivara M-C (1997) New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. Int J Numer Meth Eng 40:3313–3324
21. Hughes TJR, Franca LP, Mallet M (1987) A new finite element formulation for fluid dynamics: VI. Convergence analysis of the generalized SUPG formulation for linear time-dependent multidimensional advective-diffusive systems. Comput Methods Appl Mech Eng 6:97–112
22. Karypis G, Kumar V (1998) hMETIS 1.5: A Hypergraph Partitioning Package. Technical Report, Department of Computer Science, University of Minnesota
23. Kossaczký I (1994) A recursive approach to local mesh refinement in two and three dimensions. J Comput Appl Math 55(3):275–288
24. Kulkarni M, Pingali K, Walter B, Ramanarayanan G, Bala K, Kavita Chew LP (2007) Optimistic parallelism requires abstractions. ACM SIGPLAN Notices 42(6):211–222
25. Oliver A, Montero G, Montenegro R, Rodríguez E, Escobar JM, Pérez-Foguet A (2013) Adaptive finite element simulation of stack pollutant emissions over complex terrain. Energy 49:47–60
26. Papa DA, Markov IL (2007)*Hypergraph partitioning and clustering. Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics*, chapter 61 (2007) 61-1–61-19, CRC Press, Boca Raton
27. Paszyński M, Paszyńska A (2008) Graph transformations for modeling parallel *hp*-adaptive finite element method. Lect Notes Comput Sci 4967:1313–1322

28. Paszyński M, Paszyńska A, Grabska E (2009) Graph transformations for modeling hp-adaptive finite element method with mixed triangular and rectangular elements. Lect Notes Comput Sci 5545:864–875
29. Paszyński M, Paszyńska A, Grabska E (2008) Graph transformations for modeling hp-adaptive Finite Element Method with triangular elements. Lect Notes Comput Sci 5103:604–613
30. Pingali K, Nguyen D, Kulkarni M, Burtscher M, Hassaan MA, Kaleem R, Lee TH, Lenharth A, Manevich R, Mendez-Lojo M, Prountzos D, Sui X (2011) *The tao of parallelism in algorithms*, in Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation 12-25
31. Rivara MC (1984) Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. Int J Numer Meth Eng 20(4):745–756
32. Rivara MC (1984) Mesh refinement processes based on the generalized bisection of simplices. SIAM J Numer Anal 21(3):604–613
33. Calo VM (2005) Residual-based multiscale turbulence modeling: Finite volume simulations of bypass transition, Stanford University, Ph.D. Thesis
34. Paszyński M (2020) Klasyczna i izogeometryczna metoda elementów skończonych, AGH University, *https://epodreczniki.open.agh.edu.pl/openagh-podreczniki_view.php?categId=97&handbookId=77*
35. Paszyński M, Siwik L, Podsiadło K, Minev P (2020) A massively parallel algorithm for the three-dimensional Navier-Stokes-Boussinesq simulations of the atmospheric phenomena. Lect Notes Comput Sci 12137:102–117