Graph-grammar based longest-edge refinement algorithm for three-dimensional optimally p refined meshes with tetrahedral elements

Albert Mosiałek[1], Andrzej Szaflarski[1], Rafał Pych[1], Marek Kisiel-Dorohinicki[1], Maciej Paszyński[1], and Anna Paszyńska[2]

(1)Institute of Computer Science, AGH University of Science and Technology, Krakow, Poland
(2)Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, Krakow, Poland

Abstract. Finite element method is a popular way of solving engineering problems in geoengineering. Three-dimensional grids employed for approximation the formation layers are often constructed from tetrahedral finite elements. The refinement algorithms that avoids hanging nodes are desired in order to avoid constrained approximation on broken edges and faces. We present a new mesh refinement algorithm for such the tetrahedral grids, with the following features (1) it is a two-level algorithm, refining the elements' faces first, followed by the refinement of the elements' interiors; (2) for the face refinements it employs the graph-grammar based version of the longest-edge refinement algorithm to avoid the hanging nodes; and (3) it allows for nearly perfect parallel execution of the second stage, refining the element interiors. We describe the algorithm using the graph-grammar based formalism. We verify the properties of the algorithm, by breaking 5,000 tetrahedral elements, and checking their angles and proportions. On the generated meshes without hanging nodes we span the polynomial basis functions of the optimal order, selected via metaheuristic optimization algorithm. We use them for the projection based interpolation of formation layers.

1 Introduction

The tetrahedral three-dimensional grids are commonly employed for representation of the formation layers in geophysics [1]. There are multiple applications of the three-dimensional tetrahedral grids representing the real earth models [2,3,4,5,6]. The quantities of interest in the geophysical domain can be better approximated on the grids refined towards them. The mesh refinements algorithms can generate hanging nodes on broken edges and faces when they refine grids by breaking tetrahedral elements into smaller ones. The hanging nodes are difficult to handle, and thus the mesh refinements algorithms avoiding hanging nodes are needed. On the other hand, the broken elements must keep their proportions, to avoid approximations over elongated elements. Such the badly shaped elements i.e. with low angles generate huge numerical errors during factorization.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77970-2_16

Uniform h-adaptation might cause significant computation cost with little improvement of approximation, especially when most of the error is caused by a few elements. For this reason, the meshing algorithm should be able to generate locally dense meshes, without hanging nodes, and keeping the proportions from the initial mesh elements. Various algorithms of tetrahedral mesh refinements, avoiding the hanging nodes, and preserving the proportions have been already proposed. Most of them suggest operations on tetrahedra like bisection[7][8][9] or refinement to multiple tetrahedra at once[10][11]. Algorithms involving erasing and remeshing regions with high error have also been proposed[12][13].

The state-of-the-art algorithm for refinements of the tetrahedral elements is the longest-edge refinement algorithm proposed by Rivara [14]. The algorithm generates the path of additional refinements required to remove the hanging nodes. The parallelization of the Rivara longest-edge refinement algorithm in three-dimensions is only possible by assigning the refinement paths resulting from breaking different elements at the same time to different threads, and avoiding conflicts. Recently, we proposed a graph-grammar based version of the longest-edge refinements algorithm in two-dimensions [15]. The algorithm benefits from the graph-grammar based implementaton by a better partitioning of the computational problem into basic undividable tasks that can be executed in parallel. Comparing to the classical two-dimensional Rivara algorithm [16] it has a better parallelization potential, as described in [15]. But the three-dimensional implementation of the graph-grammar model is needed.

In this paper we propose for the first time the graph-grammar based model of the three-dimensional longest-edge refinement algorithm. It starts with executing the two-dimensional graph-grammar based longest-edge refinement algorithm on faces of the tetrahedra, taking into account the three-dimensional topology of the connected faces. Later, it breaks all the interiors of the tetrahedra, with the faces already broken in the first stage. This method allows for both better concurrency of the first stage, as shown in [15], as well as ideally parallel execution of the breaking of tetrahedral interiors in the second stage.

The structure of this paper is the following. Firstly, we briefly discuss the longest-edge refinement algorithm. Then, the definition of the hypergraph grammar is introduced. Later, we pesent the graph-grammar based model for breaking the faces of tetrahedral elements (an extension of the "flat" 2D model described in [15]). Next, we describe the pseudo-code of the graph-transformation breaking the interior of a tetrahedral with its faces already broken in the previous step. Finally, we present some numerical experiments verifying the correctness of the proposed algorithm. Namely, we break 5,000 tetrahedral elements in the mesh approximating the geological formation layers, and we monitor the hanging nodes, the angles and the proportions of the newly created elements.

2 Mesh refinement

In this section we present the definition of hypergraph grammar and all details concerning tetrahedral faces refinement.

2.1 Longest edge refinement in 2D

The longest edge refinement algorithm simply splits triangles by its longest edge. The division either creates a hanging node in an adjacent triangle or keeps the mesh conforming if the longest edge was on the edge of the mesh. Any triangles with hanging nodes are then refined again to their longest edge until there are no more triangles with hanging nodes. Note that if a triangle has a hanging node on an edge E, then it may only be refined to E or to edge longer than E. This means that hanging nodes may only be propagated to longer and longer edges. This ensures that the algorithm will eventually stop and the mesh will be conforming. However, in the worst-case scenario breaking edge E may require splitting all edges in the mesh longer than E.

2.2 Hypergraph grammar definition

We define a hypergraph as a system $G = (V, HE, t, l, a, v)$, where:
V - set of vertices
HE - set of hyperedges
t - function which maps hyperedge to sequence of vertices
l - function which maps hyperedge to label from label set C
a - function which maps vertex or hyperedge to set of attributes A
v - function which maps vertex or hyperedge attribute to value of that attribute

For our case we define $C = \{I, E, T\}$ and $A = \{x, y, z, HN, L, W, R\}$ where:
I - hyperedge label which represents interior of tetrahedron
E - hyperedge label which represents edge of tetrahedron
T - hyperedge label which represents wall of tetrahedron
$x \in \mathbb{R}$ - vertex attribute; x coordinate of vertex corner
$y \in \mathbb{R}$ - vertex attribute; y coordinate of vertex corner
$z \in \mathbb{R}$ - vertex attribute; z coordinate of vertex corner
$HN \in \mathbb{N}$ - vertex attribute; represents number of adjacent walls for which given vertex is a hanging node
$L \in \mathbb{R}$ - hyperedge attribute; represents length of the edge.
$W \in \mathbb{N}$ - hyperedge attribute; represents number of walls adjacent to given edge
$R \in \{TRUE, FALSE\}$ - hyperedge attribute; flag which denotes whether given triangle should be refined

We denote graph grammar production as $P = (LG, RG, VM, PR)$, where:
LG and RG are hypergraphs
$VM : LG \ni l \to r \in RG$ is the function which maps some veritices from LG to RG
PR - predicate of applicability. We can apply production to graph G if there exist a subgraph of G isomorphic to LG and the predicate of applicability is fulfilled. Application of production is done by substituting with RG the subgraph of graph

G isomorphic with LG so that some vertices of subgraph isomorphic to LG are replaced with corresponding vertices from RG according to VM mapping.

Hypergraph grammar is a system $HG = (G_0, PS)$, where:
G_0 is the starting hypergraph, PS is a production set

An example of tetrahedral refinement represented as hypergraph production is presented in Figure 1.

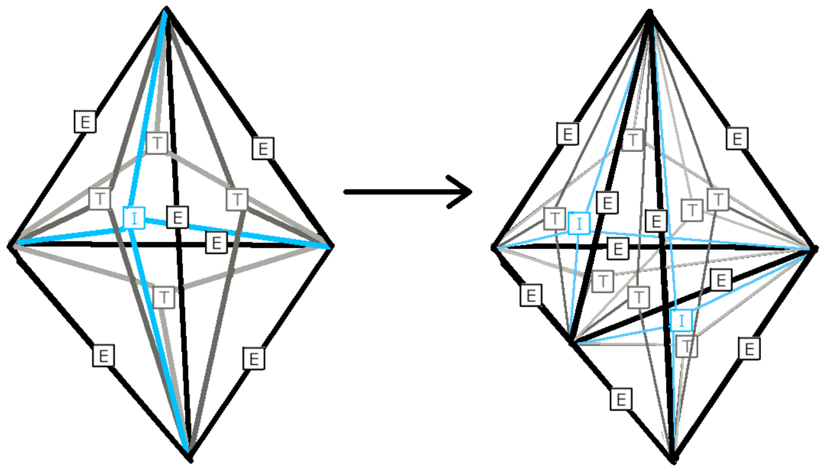

Fig. 1. Hypergraph production refining tetrahedron into two smaller tetrahedra. Right side of production has smaller labels for image clarity.

2.3 Walls refinement

In this section, we describe mesh refinement regarding walls and edges. Because of a 3D domain, our case has the following differences:

1. There are no boundary edges. Each edge have two adjacent walls on the single tetrahedron.
2. Edge might be shared between multiple tetrahedra and their walls.
3. Each vertex might be a hanging node on multiple walls.

These properties of the graph require some adjustments in graph grammar. To keep track of hanging nodes, we introduce a hanging node counter (HN) in each vertex and adjacent wall counter in each hyperedge representing a wall.

Production P1 Production P1 from Figure 2 refines element with no hanging nodes, marked with R=TRUE into two triangles. New vertex is created in the midpoint of the longest edge. It becomes a hanging node for all walls adjacent to split edge but one. Split edge keeps the number of adjacent walls. Both newly created triangles have R set to false because element is already refined. The new edge going through triangle has only two adjacent walls. Since this grammar does not support multiple hanging nodes on triangle edges, we need to ensure, that ends of the longest edge are not hanging nodes. The predicate for this production is as follows: $R(R1, L1, L2, L3, HN1, HN2) = R1$ AND $L2 \geq L1$ AND $L2 \geq L3$ AND $HN2 = 0$ AND $HN3 = 0$

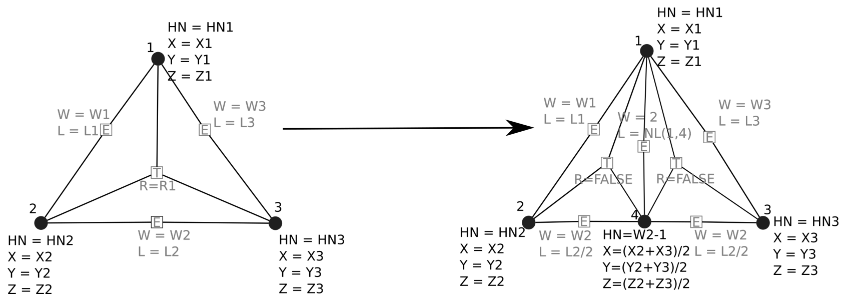

Fig. 2. Production P1.

Production P2 Production P2 from Figure 3 considers elements with one hanging node on the longest edge of the triangle. No new vertices are created. Because the vertex on the longest edge is no longer a hanging node for this wall, its HN parameter is decremented. Because the algorithm refines all elements with a hanging node, the value of R parameter does not matter in this production. We do not create any new hanging node, so we can also omit $HN = 0$ conditions in the predicate:
$R(L1, L2, L3, L4) = (L2 + L3) \geq L1$ AND $(L2 + L3) \geq L4$

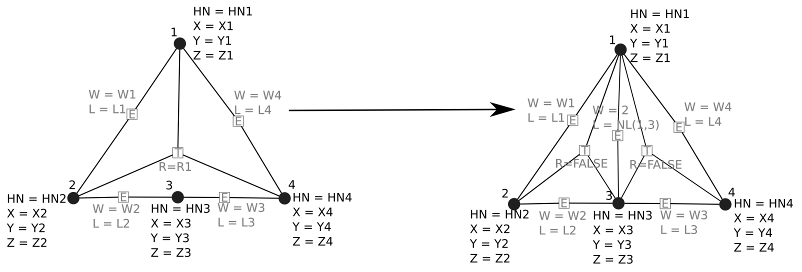

Fig. 3. Production P2.

Production P3 Production P3 from Figure 4 refines the triangle with one hanging node on other than the longest edge. A hanging node might be left for the next productions. This production works similarly to P1. The only differences are that there is strict inequality between edge to be split and edge with hanging node and refinement does not depend on the R flag. If the edges were the same length, only P2 could be applied since we prioritize reducing hanging nodes number. Predicate for P3:
$R(L1, L2, L3, L4, HN1, HN3) = L4 \geq L1$ AND $L4 > (L2 + L3)$ AND $HN1 = 0$ AND $HN4 = 0$

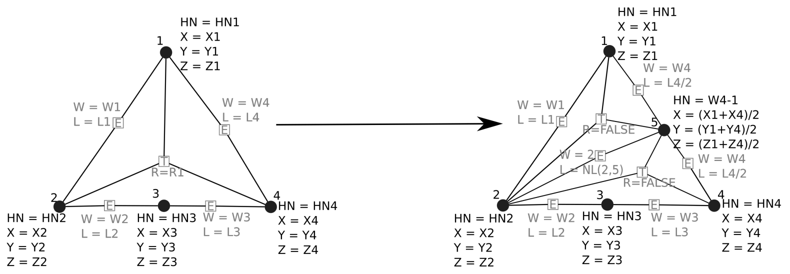

Fig. 4. Production P3.

Production P4 Production P4 from Figure 5 refines the triangle with 2 hanging nodes. One of the hanging nodes is placed on the longest edge. Both predicate and values of parameters of new hyperedge are similar to these in P2. The predicate for P4: $R(L1, L2, L3, L4, L5) = (L4 + L5) \geq (L2 + L3)$ AND $(L4 + L5) \geq L1$

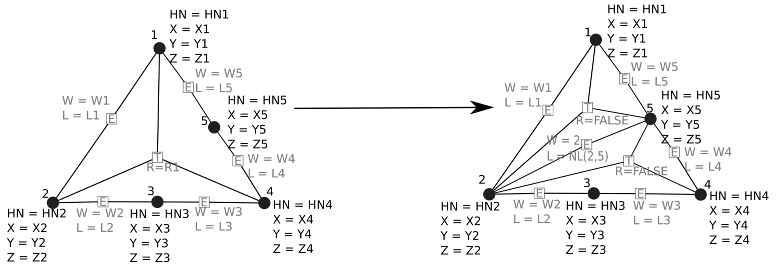

Fig. 5. Production P4.

Production P5 Production P5 from Figure 6 refines the triangle with two hanging nodes on its shorter edges. As in P3, the length of the edge has to be strictly greater than the length of the edges with hanging nodes. The predicate for P5: $R(L1, L2, L3, L4, L5, HN1, HN3) = L5 > (L1 + L2)$ AND $L5 > (L3 + L4)$ AND $HN1 = 0$ AND $HN5 = 0$

Production P6 Production P6 from Figure 7 refines the triangle with hanging node on each edge. The predicate ensures that hanging node on the longest edge will be used to split the triangles. The predicate for P6: $R(L1, L2, L3, L4, L5, L6) = (L5 + L6) \geq (L1 + L2)$ AND $(L5 + L6) \geq (L3 + L4)$

```
procedure refineTetrahedron(IEdge t):
    q := emptyQueue()
    verticesInTetrahedron := emptyList()
    tetrahedraToBeCreated := emptyList()
    //Find all vertices in the tetragedron
    Vt := getVertices(t)
    q.enqueue(Vt[0])
    verticesInTetrahedron.add(Vt[0])
    while q is not empty:
        currentVertex := q.dequeue()
        for each neighbor of currentVertex:
            if (neighbor is inside t
            AND neighbor is not in verticesInTetrahedron):
                q.enqueue(neighbor)
                verticesInTetrahedron.add(neighbor)
    //Find all tetrahedra to be created
    possibletetrahedra :=
    getAllFourElementsSubsetsOf(verticesInTetrahedron)
    for each tetrahedronVertices in possibletetrahedra:
        allVerticesAreConnected := TRUE
        for each subset in
        getAllTwoElementsSubsetsOf(tetrahedronVertices):
            if subset[0] has no edge 'E' to subset[1]:
                allVerticesAreConnected := FALSE
        if allVerticesAreConnected:
        tetrahedraToBeCreated.add(tetrahedronVertices)
    //Add missing walls and tetrahedra
    for each tetrahedronVertices in tetrahedraToBeCreated:
        for each subset in
        getAllThreeElementsSubsetsOf(tetrahedronVertices):
            if subset elements have no common 'T' hyperedge:
                CreateTEdge(subset)
        CreateIEdge(tetrahedronVertices)
    RemoveIEdge(t)
```

Listing 1.1. Tetrahedron refinement based on split walls

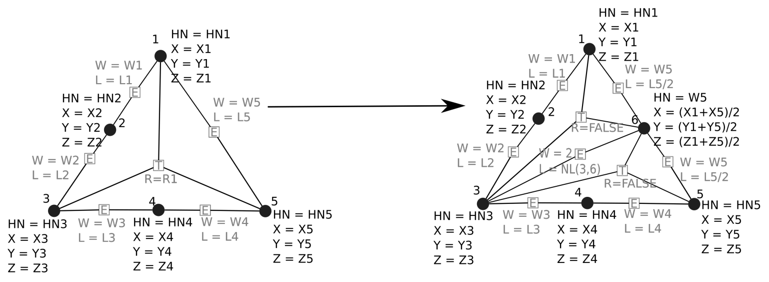
Fig. 6. Production P5.

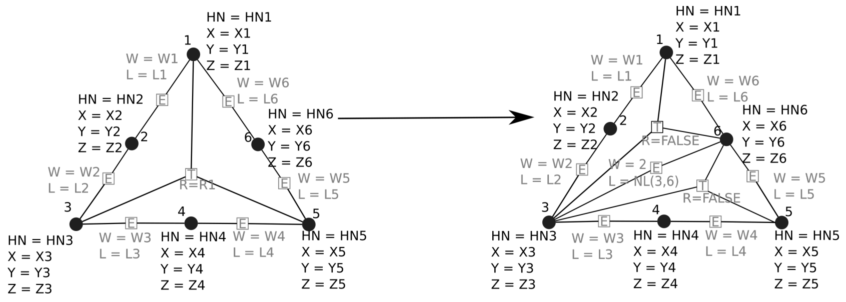
Fig. 7. Production P6.

2.4 Tetrahedron refinement

Productions described in the previous section leave tetrahedra with split walls. In this section, we present all possible outcomes of 2D refinement as well as naive pseudocode for tetrahedron refinement. We have analyzed possible results of 2D refinement for cases with broken 1-4 walls. There are 20 different cases collected in figure 11. The graph-grammar model ensures that there are no hanging nodes. From such grids with refined faces, we can break tetrahedron in a deterministic way into smaller pieces by creating new tetrahedron between any fully connected four vertices. Therefore we may treat tetrahedra represented by nets on figure 11 as left sides of hypergraph grammar production while the right side would be multiple tetrahedra generated as described above.

In listing 1.1 we present a procedure which refines tetrahedron which walls have been already split with 2D refinement. The only argument passed is the interior edge, which represents tetrahedron to be refined. lines 8-17 performs Breadth-First Search (BFS) algorithm to find all vertices which are inside the tetrahedron. More precisely all new vertices have been created on tetrahedron's walls and edges, so it is sufficient to check whether the vertex is on any of the walls of the original tetrahedron. Lines 21-30 check all combinations of 4 vertices if they are connected with each other with 'E' hyperedges. If so, new tetrahedron will be created between the chosen vertices. All new tetrahedra will contain at least one wall inside the original one, so we need to create missing walls. In lines

33-39, we add relevant hyperedges representing missing walls and tetrahedra. Finally, the original tetrahedron is removed.

3 Experimental results

In this section we verify our algorithm on the three-dimensional mesh refined towards the formation layers data. We start from the cube mesh partitioned into five tetrahedra, as presented in Figure 9. We refine faces of tetrahedra that intersects two layers, and we execute the graph-grammar based productions expressing the longest-edge refinement algorithm removing the hanging nodes on faces. Finally, we execute the graph-transformation summarized in Listing 1.1 breaking the interiors of the tetrahedra. The final mesh obtained by 5,000 refinements are presented in Figure 9. We monitor the minimal angles between edges of the created tetrahedra, as well as the ratios between the sum of the lengths of the three longest edges and the sum of all the edges, in Figure 10.

4 Projection-based interpolation metaheuristic

On each of the tetrahedral finite element we span the hierarchical basis functions following [17]. Namely, we introduce four master basis functions

$$\lambda_1(\xi_1,\xi_2,\xi_3)=1-\xi_1-\xi_2\xi_3, \quad \lambda_2(\xi_1,\xi_2,\xi_3)=\xi_1$$
$$\lambda_3(\xi_1,\xi_2,\xi_3)=\xi_2, \quad \lambda_4(\xi_1,\xi_2,\xi_3)=\xi_3$$

and we define linear basis functions, one on each element vertex

$$\psi_1(\xi_1,\xi_2,\xi_3)=\lambda_1(\xi_1,\xi_2,\xi_3) \quad \psi_2(\xi_1,\xi_2,\xi_3)=\lambda_2(\xi_1,\xi_2,\xi_3)$$
$$\psi_3(\xi_1,\xi_2,\xi_3)=\lambda_2(\xi_1,\xi_2,\xi_3) \quad \psi_4(\xi_1,\xi_2,\xi_3)=\lambda_2(\xi_1,\xi_2,\xi_3)$$

We also span p_i-1 polynomials of orders $2,...,p_i$ over element edges, $(p_m-1)(p_n-1)$ polynomials of orders $(2,2),...,(p_m,p_n)$ on element faces, and $(p_a-1)(p_b-1)(p_c-1)$ polynomials of order $(2,2,2),...,(p_a,p_b,p_c)$ on element interiors. Exemplary second order polynomials on element edges, faces and interiors are obtain by multiplications of two, three or four master basis functions

$$\psi_5(\xi_1,\xi_2,\xi_3)=\lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3)$$
$$\psi_6(\xi_1,\xi_2,\xi_3)=\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)$$
$$\psi_7(\xi_1,\xi_2,\xi_3)=\lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3)$$
$$\psi_8(\xi_1,\xi_2,\xi_3)=\lambda_4(\xi_1,\xi_2,\xi_3)\lambda_1(\xi_1,\xi_2,\xi_3)$$
$$\psi_9(\xi_1,\xi_2,\xi_3)=\lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)$$
$$\psi_{10}(\xi_1,\xi_2,\xi_3)=\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3)$$

$$\psi_{11}(\xi_1,\xi_2,\xi_3)=\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3)$$
$$\psi_{12}(\xi_1,\xi_2,\xi_3)=\lambda_1(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3)$$
$$\psi_{13}(\xi_1,\xi_2,\xi_3)=\lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3)$$
$$\psi_{14}(\xi_1,\xi_2,\xi_3)=\lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)$$

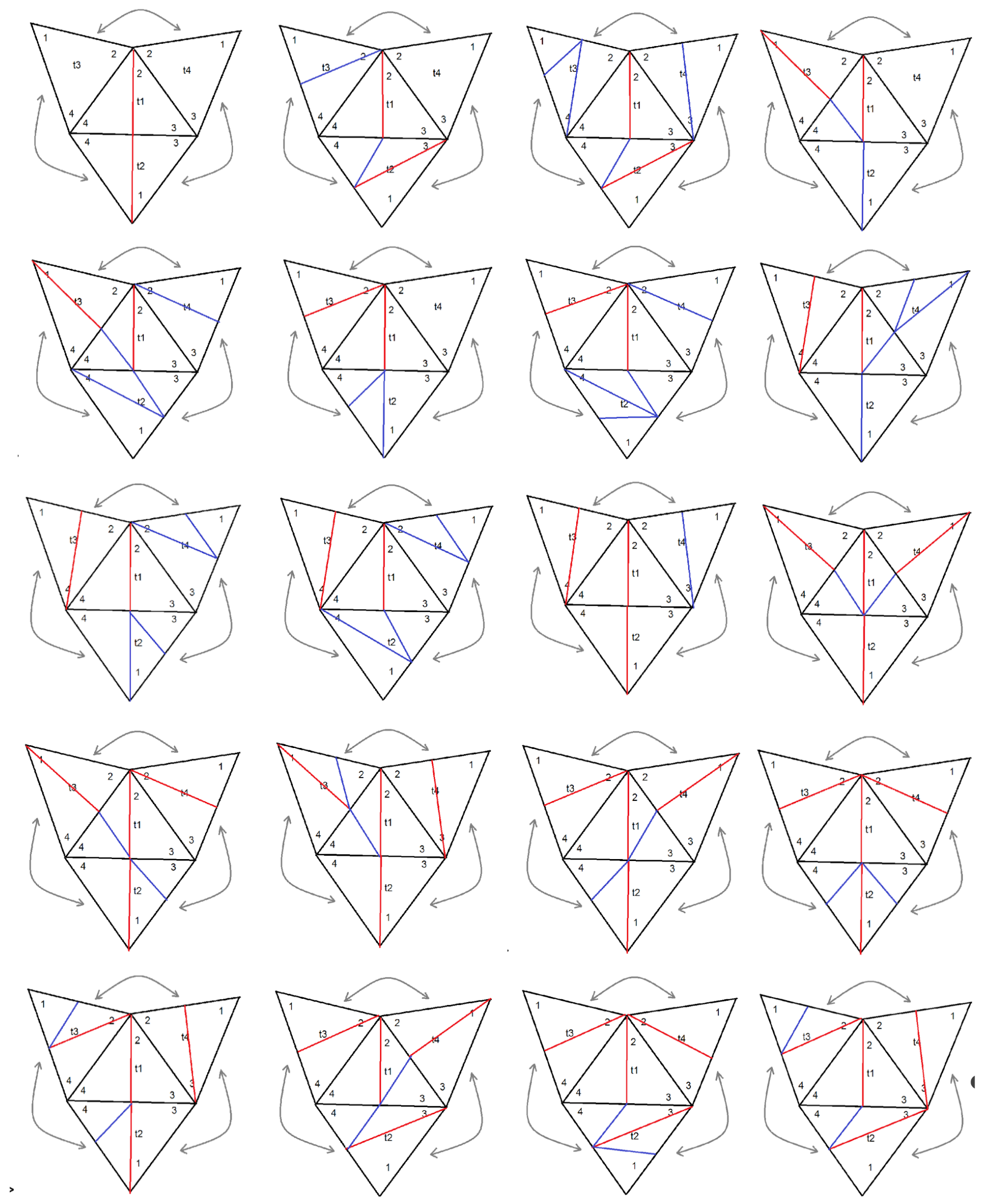

Fig. 8. Possible outcomes of 2D refinements. Red lines represents manually broken walls. Blue lines represent additional refinements done by P1-P6 productions to keep the graph consistent

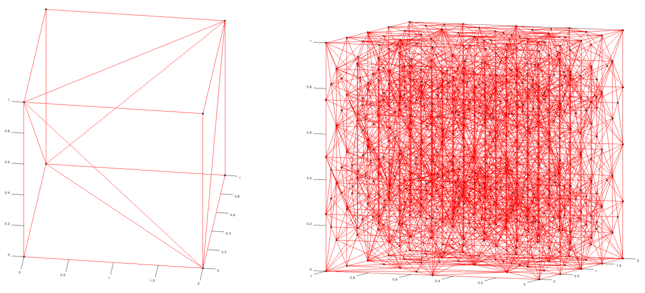

Fig. 9. Initial mesh and final mesh refined towards formation layers

$$\psi_{15}(\xi_1,\xi_2,\xi_3) = \lambda_1(\xi_1,\xi_2,\xi_3)\lambda_2(\xi_1,\xi_2,\xi_3)\lambda_3(\xi_1,\xi_2,\xi_3)\lambda_4(\xi_1,\xi_2,\xi_3)$$

For the full definition of the hierarchical basis functions we refer to [17].

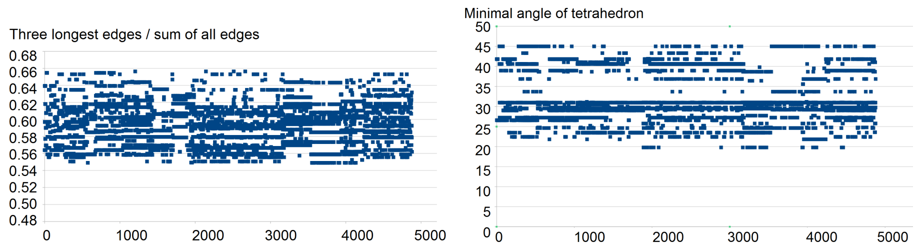

Fig. 10. Ratio between the sum of lengths of the three longest edges and the sum of all the lengths of all the edges for newly created tetrahedra (left panel), and the minimal angle between edges of the newly created tetrahedra (right panel).

Having the mesh refined towards the formation layers, we compute now the projection of the terrain data employing the projection-based interpolation algorithm. We compute coefficients for the four vertex functions at the vertices (x_i, y_i, z_i) for $i = 1, 2, 3, 4$

$$a_i = U(x_i) \quad i = 1, 2, 3, 4 \tag{1}$$

Next, we solve the L2 projection problem over each of six edges e_i

$$||(U - \sum_{i=1,\ldots,4} a_i\psi_i) - a_j\psi_j||_{L^2(e_j)} \to 0 \quad j = 5, 6, 7, 8, 9, 10 \tag{2}$$

L2 projection over four element faces f_j

$$||(U - \sum_{i=1,\ldots,10} a_i\psi_i) - a_j\psi_j||_{L^2(f_j)} \to 0 \quad j = 11, 12, 13, 14 \tag{3}$$

and finally L2 projection over element interior

$$||(U - \sum_{i=1,\ldots,14} a_i\psi_i) - a_{15}\psi_{15}||_{L^2(K)} \to 0 \tag{4}$$

We can select optimal polynomial orders of approximation at element interiors element faces, and element edges, constructing a projection of the material data. We employ the metaheuristic algorithm presented in listing 1.2, that is minimizing both numerical error and the computational cost. The algorithm is an iterative procedure. It first selects the minimal polynomial orders of approximation on finite element edges, faces and interiors. It solves the projection problem, and then it increases the polynomial order of approximation by one, in all directions, on the entire mesh. It solves the projection problem again on the finer mesh. Having the coarse $u(h, p)$ and the fine $u(h, p + 1)$ mesh solutions, it considers different refinement strategies. We can increase the polynomial order on selected edges, selected faces and over element interiors. There are several possibilities, since we have different orders in different directions. All these possibilities are considered, and we project the resulting element solution from the fine mesh into the proposed configuration of basis functions related to the considered orders configuration, given the projected reference solution w. Each refinemenet comes with the prize to pay, expressed by $dnrdof$ the number of basis functions added to implement this refinement strategy. We compute the error decrease rate $rate(w) = |u(h, p+1)-u(h, p)|-|u(h, p+1)-w|)/dnrdof$ the ratio between the error decrease and the cost. Finally, we select such the refinement, that results in the maximum error decrease rate. We proceed with these computations element wise, and for the edges and faces shared between elements we do not change the optimal orders when we consider the other element.

```
procedure SelectPRefinement(Element K,
    coarse solution u(h,p), fine solution u(h,p+1):
for mesh elements K:
    for P(i,K) different polynomial configurations over K:
        ratemin = infinity
        Compute the projection based interpolant w
            of u(h,p+1) for polynomial order P(i,K)
        Compute the error decrease rate
        rate(w)=(|u(h,p+1)-u(h,p)|-|u(h,p+1)-w|)/dnrdof
        if rate(w) < ratemin then
            ratemin = rate(w)
            Select P(opt,K) corresponding to ratemin
              as the optimal refinement for element K
```

Listing 1.2. Selection of optimal refinements over K

5 Conclusions

We proposed the graph-grammar based model of the three-dimensional longest-edge refinement algorithm for approximation of the geological formation layers.

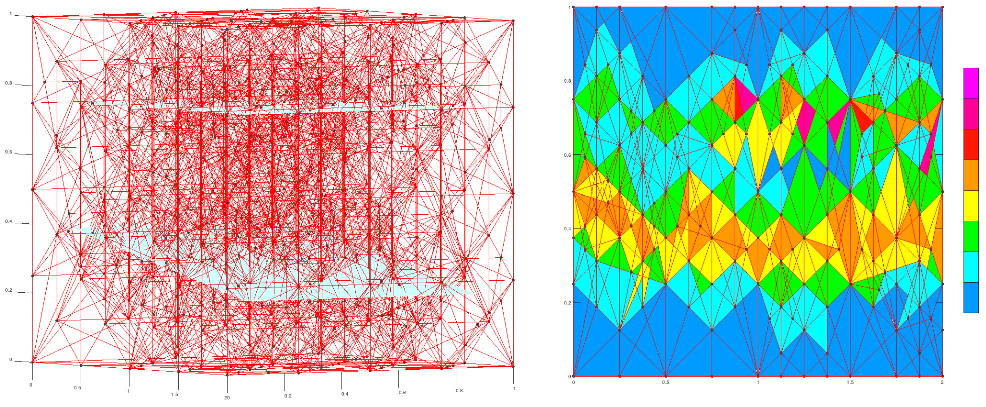

Fig. 11. Exemplary L2 projection of terrain data with three layers as well as the exemplary distribution of polynomial orders over element faces.

It starts with executing the two-dimensional graph-grammar based longest-edge refinement algorithm on faces of the tetrahedra, taking into account the three-dimensional topology of the computational mesh. In particular, it propagates the refinements between topologically adjacent faces. Next, it breaks all the interiors of the tetrahedra having some faces already broken by the first stage of the algorithm. We verified the graph-grammar based algorithm by a sequence of numerical experiments. Namely, we broke 5,000 tetrahedra towards the geological formation layers. We checked that the algorithm removed all the hanging nodes, and it preserved the proportions of the original tetrahedra. We also discussed the metaheuristic algorithm allowing for the selection of the optimal orders of approximation on element edges, faces and interiors.

6 Acknowledgement

This work was supported by National Science Centre, grant no. 2019/35/O/ST6/00571.

References

1. C.G. Farquharson, P.G. Lelièvre, S. Ansari, and H. Jahandari. Towards real earth models - computational geophysics on unstructured tetrahedral meshes? 2014.
2. Seyedmasoud Ansari and Colin G. Farquharson. *Numerical modeling of geophysical electromagnetic inductive and galvanic phenomena*, pages 669–674. 2013.
3. Peter G. Lelièvre and Colin G. Farquharson. Gradient and smoothness regularization operators for geophysical inversion on unstructured meshes. *Geophysical Journal International*, 195(1):330–341, 07 2013.

4. Peter G. Lelièvre, Colin G. Farquharson, and Charles A. Hurich. Joint inversion of seismic traveltimes and gravity data on unstructured grids with application to mineral exploration. *GEOPHYSICS*, 77(1):K1–K15, 2012.
5. V. Puzyrev, J. Koldan, J. de la Puente, G. Houzeaux, M. Vázquez, and J. M. Cela. A parallel finite-element method for three-dimensional controlled-source electromagnetic forward modelling. *Geophysical Journal International*, 193(2):678–693, 2013.
6. Christoph Schwarzbach, Ralph-Uwe Börner, and Klaus Spitzer. Three-dimensional adaptive higher order finite element simulation for geo-electromagnetics—a marine CSEM example . *Geophysical Journal International*, 187(1):63–74, 10 2011.
7. Maria-Cecilia Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21(3):604–613, 1984.
8. Douglas N Arnold, Arup Mukherjee, and Luc Pouly. Locally adapted tetrahedral meshes using bisection. *SIAM Journal on Scientific Computing*, 22(2):431–448, 2000.
9. Fernando Balboa, Pedro Rodriguez-Moreno, and María-Cecilia Rivara. Terminal star operations algorithm for tetrahedral mesh improvement. In *Lecture Notes in Computational Science and Engineering*, pages 269–282. Springer International Publishing, 2019.
10. J. Bey. Tetrahedral grid refinement. *Computing*, 55(4):355–378, December 1995.
11. Oscar Antepara, Néstor Balcázar, and Assensi Oliva. Tetrahedral adaptive mesh refinement for two-phase flows using conservative level-set method. *International Journal for Numerical Methods in Fluids*, 93(2):481–503, August 2020.
12. Célestin Marot, Jeanne Pellerin, and Jean-François Remacle. One machine, one minute, three billion tetrahedra. *International Journal for Numerical Methods in Engineering*, 117(9):967–990, 2019.
13. Wei Guo, Yufeng Nie, and Weiwei Zhang. Parallel adaptive mesh refinement method based on bubble-type local mesh generation. *Journal of Parallel and Distributed Computing*, 117:37–49, 2018.
14. María-Cecilia Rivara. Local modification of meshes for adaptive and/or multigrid finite-element methods. *Journal of Computational and Applied Mathematics*, 36(1):79–89, 1991. Special Issue on Adaptive Methods.
15. Krzysztof Podsiadło, Albert Oliver, Anna Paszynska, Rafael Montenegro, Ian Henriksen, Maciej Paszynski, and 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*, 12 2020.
16. Maria-Cecilia Rivara. Mesh refinement processes based on the generalized bisection of simplices. *SIAM Journal on Numerical Analysis*, 21(3):604–613, 1984.
17. Leszek Demkowicz, Jason Kurtz, David Pardo, Maciej Paszynski, Waldemar Rachowicz, and Adam Zdunek. *Computing with hp-adaptive finite element method, Volume II Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications.* Taylor & Francis, CRC Press, 2008.