Journal of Computational Science 64 (2022) 101856

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Graph-grammar based algorithm for asteroid tsunami simulations

Paweł Maczuga [a], Albert Oliver-Serra [b], Anna Paszyńska [c], Eirik Valseth [d,e], Maciej Paszyński [a,*]

[a] AGH University of Science and Technology, Poland
[b] The University of Las Palmas de Gran Canaria, Spain
[c] Jagiellonian University, Kraków, Poland
[d] The Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, USA
[e] University of Oslo, Norway

ARTICLE INFO

Keywords:
Graph grammar
Longest-edge refinement algorithm
Finite element method
Non-linear wave equation
Scientific computing in Julia

ABSTRACT

On January 18, 2022, around 1 million kilometers from Earth, five times the distance from Earth to the Moon, a large asteroid passed without harm to the Earth. Theoretically, however, the event of the asteroid falling into Earth, causing the tsunami, is possible since there are over 27,000 near-Earth asteroids [1], and the Earth's surface is covered in 71 percent by water. We introduce a novel graph-grammar-based framework for asteroid tsunami simulations. Our framework adaptively generates the computational mesh of the Earth model. It is built from triangular elements representing the seashore and the seabed. The computational mesh is represented as a graph, with graph vertices representing the computational mesh element's interiors and edges. Mesh refinements are often performed by the longest-edge refinement algorithm. We have expressed this algorithm by only two graph-grammar productions. The resulting graph represents the terrain approximating the topography with a prescribed accuracy. We generalize the graph-grammar mesh refinement algorithm to work on the entire Earth model, allowing the generation of the terrain topography, including the seabed. Having the seashore and the seabed represented by a graph, we introduce the finite element method simulations of the tsunami wave propagation. We illustrate the framework with simulations of the disastrous asteroid falling into the Baltic sea.

1. Introduction

In this paper, we introduce a graph-grammar-based platform for performing tsunami simulations. There are the following novelties of our approach

- Using a novel graph-grammar-based approach, we adaptively generate the computational mesh covering the coastal area and the seabed. For the first time, we employ the "adjoint" graph, where graph vertices represent the interiors and edges of the triangular elements. Our previous model [2] used the "direct" representation, where graph vertices represent element vertices, and graph edges represent the element edges and interiors. It required costly identification of the common edges of triangles. The element interiors were connected with element vertices, and thus it required identification of common edges by checking neighbors of the vertices, possibly considering hundreds of adjacent edges. In this paper, we consider the "adjoint" graph, and we connect the edges with interiors; and since each edge always has two vertices, the identification is straightforward.
- We propose two graph transformations to express the longest-edge mesh refinement algorithm [3,4]. Thus, our model is a generalization and simplification of the graph-grammar-based model of the Rivara algorithm [2], which employed six graph-grammar productions for modeling of longest-edge refinements.
- We formulate our graph-grammar-based model in spherical coordinates, and we execute the generation of the seabed and seashore for the entire Earth model.
- We employ a finite element method as well as an explicit time integrator and the graph-grammar-based meshes in simulations of tsunami propagation. For this purpose, we use a hyperbolic wave equation from [5]. Furthermore, we present verification of our numerical procedure on a hyperbolic model problem.
- To showcase the developed framework, we present a simulation of an asteroid tsunami hitting the Baltic sea.

In this paper, we use the concept of graph grammar. In our previous work, we used hypergraph grammar, initially introduced by Habel and Kreowski in [6,7] for applications in computer graphics. Special

* Corresponding author.
E-mail addresses: eiriva@math.uio.no (E. Valseth), maciej.paszynski@agh.edu.pl (M. Paszyński).

https://doi.org/10.1016/j.jocs.2022.101856
Received 28 May 2022; Received in revised form 19 August 2022; Accepted 4 September 2022
Available online 17 September 2022

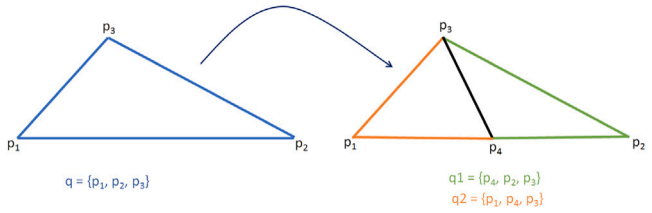


Fig. 1. Longest edge refinement of the triangular element.

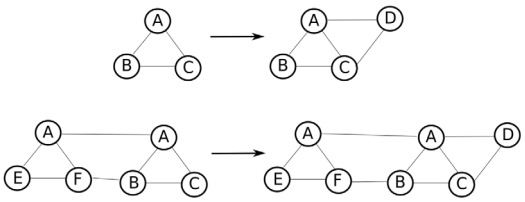


Fig. 2. First row: single graph-grammar production. Second row: application of the graph-grammar production.

algorithms are also developed and optimized for the hypergraphs [8–10]. This work is based on our experience with exploring the graph-grammar-based simulations developed for the air pollution simulations in the Lesser Poland district area [2]. Here, we propose a longest-edge refinement algorithm with a smaller number of productions; with cheaper identification of the left-hand sides of the productions, we formulate the model in the spherical system of coordinates. We also implement a non-linear wave equation, and employ our method in asteroid tsunami simulations.

There exist several alternative codes for tsunami simulations, see, e.g., [11,12]. The software presented in [11] performs local tsunami simulations in nearshore areas. Experimental results from [11] demonstrated there show 2 h long computations using four cores of the simulations (based on the OpenMP loop parallelization), equivalent to 8 h of single-core computations. In this paper, we present an alternative graph-grammar-based model. The model decomposes the computational problem into basic tasks called graph-grammar productions. They can be employed as the model of concurrency, with particular graph-grammar productions executed in parallel (after implementation in, e.g., GALOIS [13] or Katana-graph [14] graph transformation systems). In [12] the formulation of the shallow water equations in the spherical coordinates system is presented along with a family of high-order computational methods. However, they do not present large-scale simulations; they focus on local Mediterranean tsunami simulations.

The expression of a tsunami simulator using the graph-grammar model, as proposed herein will allow us to build a concurrency model. In turn, this allows us to identify graph grammar productions as basic undividable tasks that can be executed concurrently to perform large parallel simulations of tsunami propagation in global Earth models. In this paper, we focus on Cartesian coordinate system formulation, suitable for local Baltic sea simulations, and we develop the optimized graph-grammar model and test it using sequential implementation and explicit model.

The structure of the paper is as follows: Section 2 summarizes the longest-edge refinement algorithm; Section 3 describes how to express the algorithm by two graph productions working on the "adjoint" graph; Section 4 describes the algorithm's execution to generate the computational mesh covering Earth; in Section 5, generate a computational mesh covering the Baltic seashore and seabed; Section 6 describes the numerical formulation of the non-linear wave equation solver in Cartesian coordinates, suitable for the local phenomena simulations; we verify the solver on a model problem in Section 7; in Section 8, we present a numerical experiment of asteroid tsunami simulations in the Baltic sea; Section 9 presents the comparison of the computational costs of identification of the left-hand sides of the graph-grammar productions in our new model versus the previous six productions model; finally, in Section 10 we conclude the paper.

2. Longest-edge refinement algorithm

Mesh refinement is task in which elements of a mesh are subdivided to obtain a finer mesh. During such mesh refinement processes, the original nodes are not removed, and the topology of the original mesh is preserved. Usually, we refine the elements where the error is large in order to compute a better numerical solution. Here, we will consider the longest-edge refinement algorithm. This algorithm always breaks the element along its longest edge, see Fig. 1 for a visualization. The longest-edge refinement can be expressed mathematically as the bisection of a simplex $q = (p_1, p_2, \dots, p_n, p_{n+1}) \in R^n$. If the distance between p_k and p_m is the maximum distance of the simplex, then a new point is created such that $p = \frac{p_k + p_m}{2}$ and two new simplices are created as: $q_1 = (p_1, p_2, \dots, p_{k-1}, p, p_{k+1}, \dots, p_m, \dots, p_{n+1})$ and $q_2 = (p_1, p_2, \dots, p_k, \dots, p_{m-1}, p, p_{m+1}, \dots, p_{n+1})$. Hence, mathematically the longest-edge can be applied in any dimension. Geometrically, the longest edge refinement generates a new point in the middle (a hanging node) of the longest edge and generates the new two elements.

The main idea is to break an edge in half only if it is the longest edge of all adjacent elements (we call this edge a terminal edge). For this purpose, when a triangle t is marked for refinement, we must consider all neighboring elements until we find a terminal edge. All transition elements are known as the Longest Edge Propagation Path $LEPP(t)$. The algorithm divides the last two elements of the $LEPP(t)$ (or the last single element located on the boundary) and reconstructs it again until $LEPP(t)$ is empty. This method ensures conformity of the mesh (i.e., no hanging nodes). We do not know beforehand how many additional elements need to be broken to ensure the conformity of the mesh and it may require several refinements of a single element. The traditional algorithm is presented below in Algorithm 1.

Algorithm 1 Longest-Edge-Refinement

Require: *t triangle to refine, T mesh of triangular elements*

```
while t remains without being bisected do
    Find LEPP(t)
    t* = the last triangle of LEPP(t)
    if t* is a terminal boundary triangle then
        bisect t*
    else
        bisect the last pair of terminal triangle of LEPP(t)
end while
```

3. Expressing the longest-edge refinement algorithm by graph-grammar productions

We propose to represent the mesh by a graph and the mesh generation and refinements by so-called graph grammar productions. Graph grammar productions consist of two graphs: the left-hand-side graph and the right-hand-side graph. It allows us to generate new, more complex graphs from simple graphs by replacing the subgraph of the simple graph isomorphic to the left-hand-side graph with the right-hand-side graph, see Fig. 2. The main disadvantage of graph grammar is the computational cost of finding the graph isomorphic with the left-hand-side graph in generated graph.

In our approach, a triangular element is represented as a graph, as shown in Fig. 3, where the interior of the triangle is denoted by the label T, and the edges are denoted by the label E. Additionally, each triangle edge remembers the pointer to its beginning and endpoint (nodes). This representation aims to reduce the computational cost of finding the left-hand side. In our implementation, we keep pointers

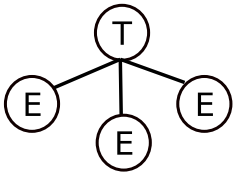


Fig. 3. Graph representation of a triangle.

to the interiors of elements, and from the interior node, we identify the edges of the element. This method is better than connecting the interiors and vertices of an element as this negates the need to check all edges attached to the vertices while looking for the subgraph representing one particular triangular element. Each triangle node T is augmented with a single attribute:

- R: The triangle is marked to be refined

Whereas the edge nodes E are augmented with the following attributes:

- LE: The edge is one of the longest-edges.
- BR: The edge is broken.
- AE: Number of adjacent elements (1 if is a boundary, 2 if is interior).
- `x, y, z`: Coordinates of the edge (middle point).
- IP: Pointer to the initial point.
- FP: Pointer to the final point.

To simplify the applicability predicate, use a deterministic comparison, and make the transition to 3D smoother, we introduce the following comparison operator:

```
1  bool LESS(e1, e2)
2  {if (!(e1.AE == e2.AE) {return e1.AE < e2.AE;}
3  if (!equal(e1.x, e2.x) {return e1.x < e2.x;}
4  if (! equal(e1.y, e2.y) {return e1.y < e2.y;}
5  if (! equal(e1.z, e2.z) {return e1.z < e2.z;}}
```

Listing 1.1: Comparison operator LESS.

where e1 and e2 are edges, $LESS(v1, v2)$ denotes the two argument Boolean operator comparing two edges, and $LE(v1)$ denotes the single argument Boolean function checking if $v1$ is the longest edge.

$LESS(e1, e2)$ will return true if:

- e1 is the boundary, and e2 is not on the boundary of the mesh, or
- (both edges are boundary edges, or both are interior edges), and x coordinate of e1 is smaller than x coordinate of e2, or
- (both edges are boundary edges, or both are interior edges) and x coordinates of e1 and e2 are the same and y coordinate of e1 is smaller than y coordinate of e2, or
- (both edges are boundary edges or both are interior edges) and x and y coordinates of e1 and e2 are the same, and z coordinate of e1 is smaller than z coordinate of e2. In all other cases, it returns false.

This operator is used for defining the productions in such a way that the boundary edges will get broken first, and if there are two broken or two unbroken edges in a triangle, an edge with smaller values of coefficients will be broken first.

We describe the process of mesh refinements by two graph transformations, called graph-grammar productions. These expand the sub-graph provided on the left-hand side of the production by the sub-graph described on the right-hand side. The number of left-hand sides is 2 and it depends on whether the "longest-edge" is broken or not, see Fig. 4. This also ensures that the number of productions is as small as possible. Production 0 is executed before the longest-edge refinement algorithm

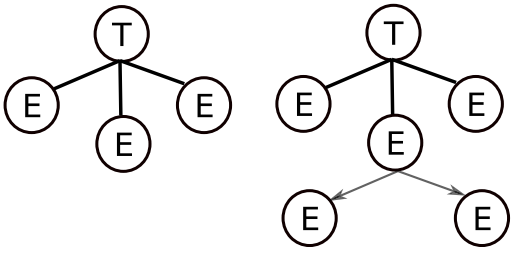
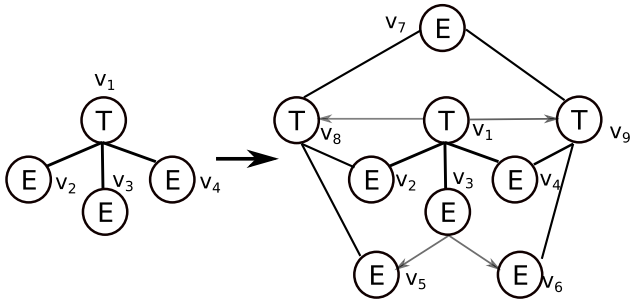


Fig. 4. Two possible left-hand sides: unbroken edge and broken edge.



(NOT BR(v_3) AND LE(v_3) AND

(R(v_1) OR BR(v_2) OR BR(v_4)) AND

(NOT (BR(v_2) AND LE(v_2))) AND

(NOT (BR(v_4) AND LE(v_4))) AND

(NOT (NOT BR(v_2) AND LE(v_2) AND LESS(V_3,V_2))) AND

(NOT (NOT BR(v_4) AND LE(v_4) AND LESS(V_3,V_4)))

Fig. 5. Graph-grammar production (P1) with predicates of applicability.

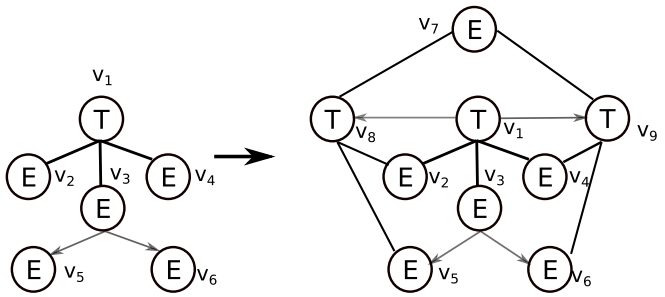
starts and sets the refinement flag for an element. This only changes the value of the attribute R of the node labeled by T to *true*. Production 1 bisects the triangle by an edge that is not broken (see Fig. 5) and is executed if:

- Edge v_3 is not broken, and is one of the longest edges, described by $NOT\,BR(v_3)\ AND\ LE(v_3)$.
- The element has to be refined, it is marked to be refined, or it has one edge broken, denoted by $R(v_1)\ OR\ BR(v_2)\ OR\ BR(v_4)$.
- If edge v_2 is broken, it cannot be one of the longest-edges. If edge v_3 is broken, it cannot be one of the longest-edges. This is described by $NOT\ (BR(v_2)\ AND\ LE(v_2)$ and $NOT\ (BR(v_4)\ AND\ LE(v_4))$,
- If edge v_2 is not broken but it is one of the longest edges, check the comparison operator. Same for edge v_4. This is described by $NOT\ (NOT\ BR(v_2)\ AND\ LE(v_2)\ AND\ LESS(V_3, V_2))$ and $NOT\ (NOT\ BR(v_4)\ AND\ LE(v_4)\ AND\ LESS(V_3, V_4))$.

New graph nodes represent the newly created edges. They are sons of the father node, representing the broken edge. Production 2 bisects the triangle by a broken edge, as shown in Fig. 6. The production can be executed if:

- Edge v_3 has to be one of the longest, described by $LE(v_3)$.
- If edge v_2 is also broken and is one of the longest edges, check the comparison operator. Same for edge v_4. This is described by $NOT\ (BR(v_2)\ AND\ LE(v_2)\ AND\ LESS(V_3, V_2))$ and $NOT\ (BR(v_4)\ AND\ LE(v_4)\ AND\ LESS(V_3, V_4))$.

Fig. 7 presents the control diagram guiding the application of graph grammar productions. First, production $P0$ is to be performed. If $P0$



LE(v_3) AND

(NOT (BR(v_2) AND LE(v_2) AND LESS(V_3,V_2))) AND

(NOT (BR(v_4) AND LE(v_4) AND LESS(V_3,V_4)))

Fig. 6. Graph-grammar production (P2) with predicates of applicability.

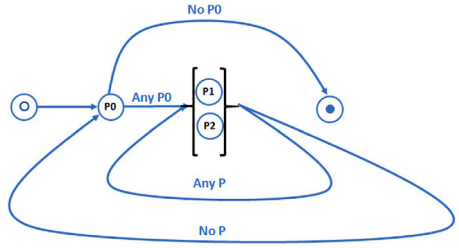


Fig. 7. Diagram controlling the execution of graph-grammar productions.

was not performed, all elements have the required approximation quality, and we go to the final state of the diagram. If production $P0$ was performed, $P2$ and $P1$ are to be performed (in a loop). If performed one or more times, $P1$ and $P2$ cannot be performed, again, we move to $P0$.

4. Adaptive generation of the earth topography model, including the terrain and the seabed

We present the graph-grammar-based derivation of the global model of the topography of Earth. Our goal is to generate a mesh of a sphere representing the Earth. We first create an initial, low-resolution mesh of a sphere and then adapt it using the longest-edge refinement algorithm expressed by two graph-grammar productions. The initial mesh is already a complete sphere, including poles. It is represented as a graph as described before, but each vertex has two types of coordinates

- x, y, z - Cartesian coordinates of a vertex on a sphere.
- u, v - projection of a vertex to flat surface in form of spherical coordinates, where $u \in [-180°, 180°)$ and $v \in [-90°, 90°]$. In other words u is longitude and v is latitude of the vertex.

All vertices are added using u, v coordinates and then translated to x, y, z, using the following equations:

$$x = r\cos(u)\cos(v),$$

$$y = r\sin(u)\cos(v),$$

$$z = r\sin(v),$$

where r denotes the distance in the direction of the radius of Earth.

4.1. Initial mesh

We start from rectangular mesh (in u, v coordinates) partitioned into triangular elements, without poles, and one horizontal segment that contains $u = 180°$, with vertices uniformly spaced. Assuming that we want the sphere to contain n horizontal segments and m vertical ones, this pre-initial mesh will have $n - 1$ and $m - 2$ segments, where coordinates range from $(-180°, -90° + \frac{180°}{m})$ in the bottom-left corner to $(180° - \frac{360°}{n}, 90° - \frac{180°}{m})$ in the upper-right corner, see Fig. 8. In the resulting mesh, the leftmost vertices with $u = -180°$ are, in fact, the same as the missing 180° vertices on the right. We fill in the missing segment by connecting corresponding vertices in order to have a proper poles-less sphere in 3D, as shown in Fig. 9. Lastly, we add two new vertices for poles in coordinates $(0°, 90°)$ and $(0°, -90°)$ along with appropriate triangles as shown in Fig. 10.

4.2. Longest-edge refinement of a sphere mesh

Ultimately we use the longest-edge refinement algorithm on the generated initial mesh to create a high-resolution mesh that may be used in simulations. The algorithm operates on single triangles and there are two issues to address:

- How is the distance between two vertices calculated? The graph-grammar-based longest-edge algorithm will employ this distance to determine elements to be broken.
- How and where to add new vertices after breaking an edge in spherical coordinates?

The distance can be calculated using either x, y, z or u, v coordinates. However, using u, v introduces two problems: (1) connecting the left-most and rightmost triangles, and (2) single vertices at the poles (see Fig. 10). In the end, the distance should be calculated as in the standard latitude–longitude projection: Assume we have two vertices with coordinates (u_1, v_1) and (u_2, v_2). We calculate distance as a Cartesian distance but introduce two additional conditions:

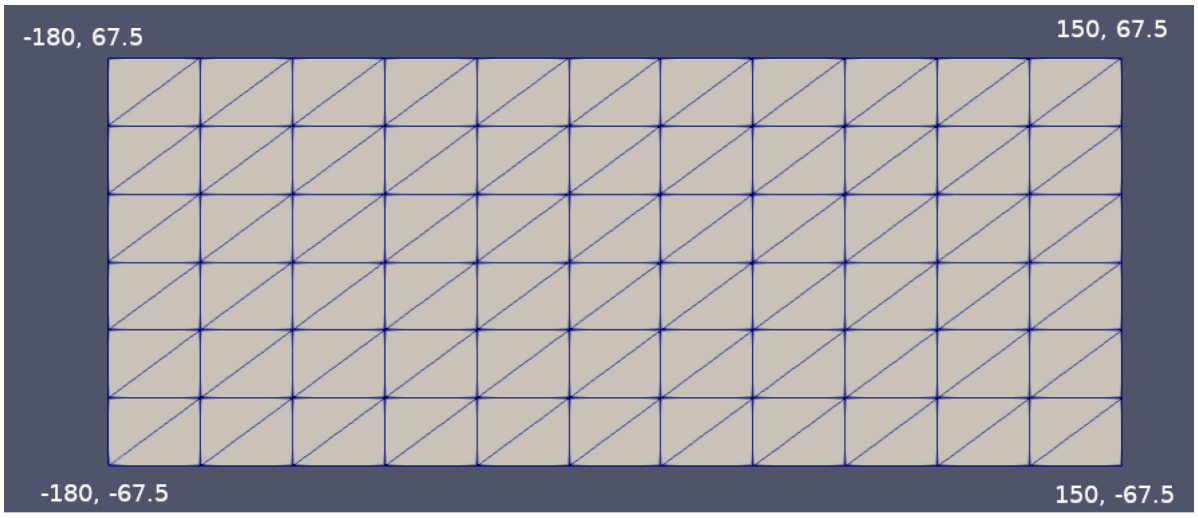
- If $u_1 = -180°$ and $u_2 > 0°$ we assume that $u_1 = 180°$.
- If $|v_1| = 90°$ (north or south pole) we assume that $u_1 = u_2$.

A new vertex with coordinates (u_3, v_3) is added in the middle of the broken edge. Again there are two exceptions - poles and $u = -180°$:

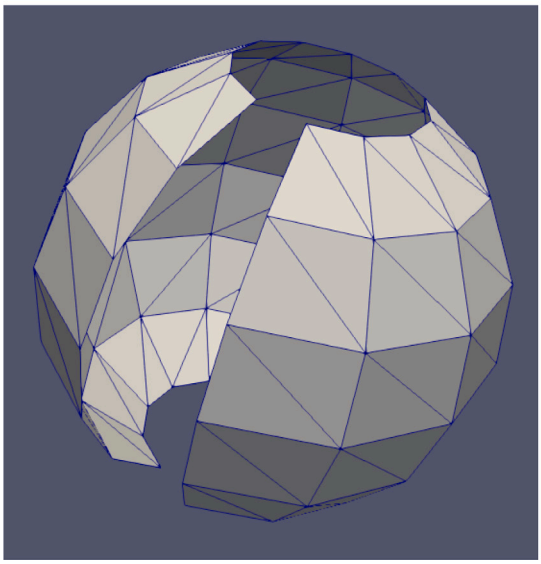
- If $u_1 = -180°$ and $u_2 > 0°$ then $u_3 = \frac{180° + u_2}{2}$.
- If $|v_1| = 90°$ then $u_3 = \frac{u_2 + u_2}{2} = u_2$.

Selection either u, v or x, y, z for calculating distance will produce different results, especially near the poles. The finer the initial grid is, the more similar meshes will be near the equator. The result for 8 iterations of adaptation is shown in Fig. 11 for x, y, z and Fig. 12 for u, v. Every triangle is marked for refinement at each iteration, and the longest-edge refinement algorithm is used on the mesh.

Finally, we execute the longest-edge refinement algorithm expressed by graph-grammar productions to create the mesh of the entire Earth using elevation data from the GMRT database [15]. As the initial mesh, we use the coarse sphere with the elevation of the vertices adjusted using data from [15], with 143 vertices, 264 triangles, and 407 edges. Since the database does not contain data for the poles, we cut the sphere's top and bottom at 65° and −65° degrees. The resulting mesh is shown in Fig. 13, where we have executed 13 iterations of the longest-edge refinement algorithm expressed by the control diagram. The final mesh consists of 404,912 vertices, 808,692 triangles, and 1,213,604 edges.



(a) Initial mesh in u, v coordinates.



(b) Initial mesh in x, y, z.

Fig. 8. Initial mesh of the entire sphere without poles and single vertical segment. Full mesh with all segments will have 12 vertical ones and 8 horizontal.

5. Adaptive generation of the baltic sea topography

In this section, we focus on a detailed local model, where we will perform the tsunami simulations. Namely, we present an exemplary derivation of the topography of the Baltic seashore and seabed. We start from a graph representing four initial triangular elements; see Fig. 14. We adjust the height of the mesh's triangles to the terrain height using topographic data from the same database as previously [15] including seashore and seabed. We iteratively break elements where the approximation is poor. To speed up the generation process, we employ the following method for identification of the poor-quality triangles: we identify all points from the map located within the triangle where we have the height data available. For each point, we compute the absolute value of the distance between the height from the map and the height as approximated by the triangle. We mark this triangle for refinement if an absolute value for any point is larger than a prescribed tolerance. Additional mesh refinements may be needed as required by the longest-edge refinement method to ensure there are no hanging nodes in the mesh. The resulting iterations are presented in Figs. 15–22, where the final mesh (see Fig. 23) has 217,058 nodes, 433,101 triangles, and 650,158 edges.

6. Tsunami simulations with nonlinear wave equations in shallow water

In this section, we introduce our modeling approach for simulations of tsunami waves. Tsunami wave propagation is a result of seawater flow induced by rapid shifts in the sea bottom [5] or by sea surface impacts of large objects from, e.g., rock slides [16] or asteroids [17]. Such seawater flow is governed by shallow water equations [18]. These highly nonlinear partial differential equations are intractable and typically require significant computational resources for physically relevant and accurate solutions. However, the nature of tsunami wave flows allows us to consider greatly simplified models with sufficient fidelity for the sea water flows. Here, we follow the approach taken in [5,19] and model the tsunami propagation using a nonlinear wave equation:

$$\frac{\partial^2 u}{\partial t^2} - \nabla \left(g(u - z) \nabla u \right) = 0, \tag{1}$$

where u denotes water surface elevation, z the terrain topography and bathymetry, and $g = 9.81 \frac{m}{s^2}$ the constant of gravitational acceleration.

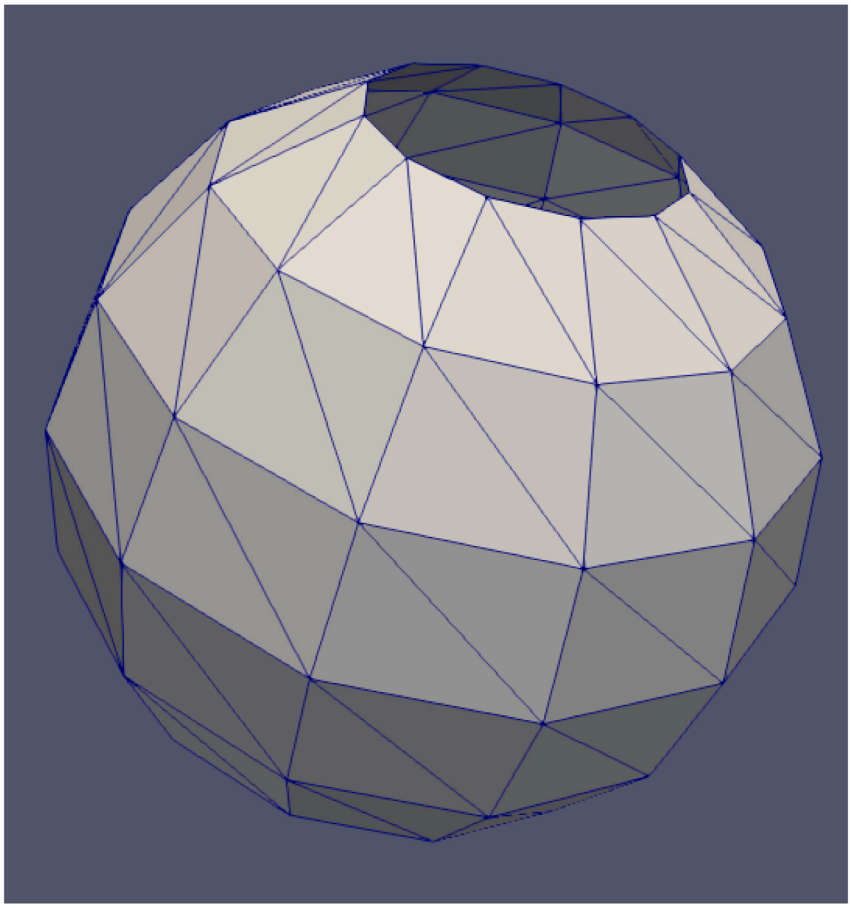


Fig. 9. Initial mesh with all horizontal segments.

We note that there more simplified shallow water models in existence, including fully linear models, e.g., those based on Boussinesq approximations [20].

To compute approximations of the wave equation (1), we apply a Bubnov–Galerkin [21] finite element method for the spatial discretization and an explicit finite difference time stepping scheme for temporal discretization. First, we introduce a finite difference approximation of the second time derivative $\frac{\partial^2 u}{\partial t^2} \approx \frac{u_t - 2u_{t-1} + u_{t-2}}{dt^2}$. We subsequently employ it in an explicit time integration scheme:

$$\frac{u_t - 2u_{t-1} + u_{t-2}}{dt^2} = \nabla \left(g(u_{t-1} - z) \nabla (u_{t-1}) \right) = 0. \tag{2}$$

Next, we collect the unknowns on the left-hand side:

$$\underbrace{u_t}_{\text{Next state}} = \underbrace{u_{t-1}}_{\text{Previous state}} + \underbrace{u_{t-1} - u_{t-2}}_{\text{States difference}} + \underbrace{\Delta t^2}_{\text{Time step squared}} \underbrace{\nabla \left(g(u_{t-1} - z) \nabla u_{t-1} \right)}_{\text{Physics}}. \tag{3}$$

To apply the Bubnov–Galerkin method, we multiply by test functions v, integrate the spatial derivative term by parts and apply the following boundary conditions on the whole domain boundary

$$\nabla (g(u_t - z)) \cdot n = 0 \tag{4}$$

which are equivalent to $\nabla (u_t - z) \cdot n = 0$ and in case of $z = const$ they reduce to zero Neumann boundary conditions $\nabla u_t \cdot n = 0$. We get:

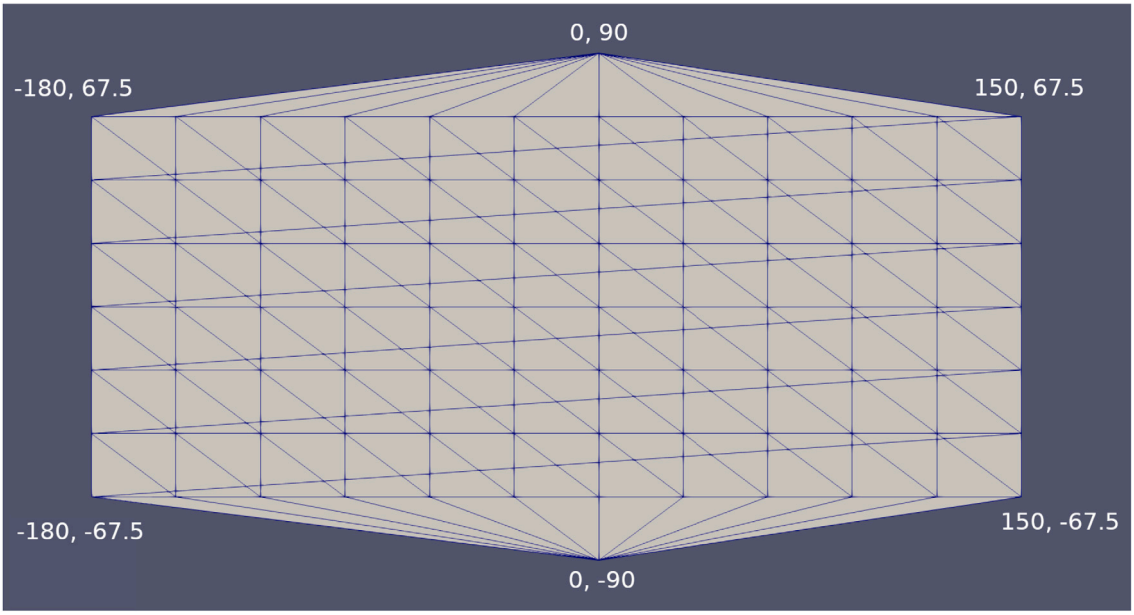
$$\begin{aligned} &\text{Find } u \in V : \left(u_t, v \right) = \left(u_{t-1}, v \right) + C \left(u_{t-1} - u_{t-2}, v \right) \\ &\quad - \Delta t^2 \left(g(u_{t-1} - z) \nabla u_{t-1}, \nabla v \right) \quad \forall v \in V. \end{aligned} \tag{5}$$

In (5), we have introduced a damping constant C in front of the wave propagation term. In the full nonlinear shallow water equations, see [18], there are complex sources, including sea bed and sea surface friction. In the simplified wave model, we consider these terms are absent, but we represent them by the damping constant. In the Bubnov–Galerkin finite element method, we use triangular finite elements and linear basis functions related to the vertices of the computational mesh. We have developed our original finite element method solver in Julia [22].

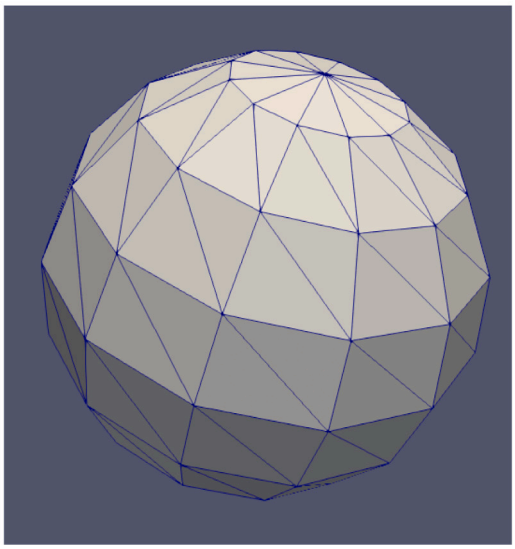
7. Experimental verification on the model problem

As we mentioned in the previous section, our wave equation is an approximation to the shallow water equations. In the resulting approximation, we introduce the damping constant as a mechanism to account for the simplifications. In real-life phenomena simulations, the damping constant can be obtained experimentally in order to match the measurements of the real tsunami phenomena. In this section, we show that our non-linear wave equation with $C = 1.0$ (without the damping constant) matches the exact solution in the idealized "swimming pool" case without friction. We also illustrate how changing this dumping constant influence the wave shape. Thus, we consider a simplified case in which we have an exact closed-form analytical solution to our model problem as described in [5]. Thus, our numerical experiment uses:

- The analytical solution from [5] (denoted as the "exact" in the plots)
- A uniform mesh of 32,764 triangular elements, which covers the square region $(0, 10) \times (0, 10)$.
- Initial condition : $u_0(r) = 2e^{-r^2} + 2$ where r denotes the distance from the center point $(5, 5)$



(a) Full initial mesh in u, v. Note that since each vertex represents one and exactly one vertex in the 3D space, it is impossible to project this mesh to a flat surface without overlapping. The rightmost vertices are connected to the leftmost ones to form proper triangles in 3D. Furthermore, contrary to latitude-longitude projection, each pole is a single vertex.



(b) Full initial mesh in x, y, z.

Fig. 10. Full initial mesh counting all the segments and poles.

– zero-Neumann kind b.c. in the model "pool case" $\nabla(g(u_t - z)) \cdot n = 0$ which reduces to the zero Neumann b.c. $\nabla u_t \cdot n = 0$, meaning there is no flow being pushed in and out (we are in the closed swimming pool).

We ran the simulation with time step $dt = 0.001$ s, and a final time 2 s.

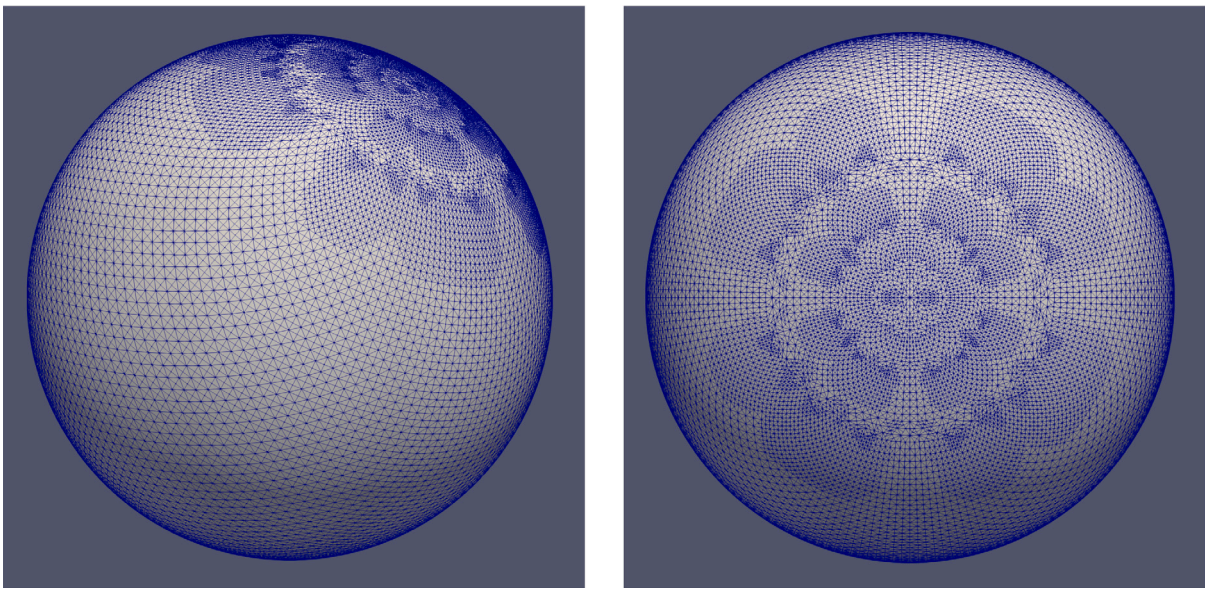
In Fig. 24, we present surface plots of the waves for selected times. As the initial wave travels towards the boundary it is reflected and self interacts as expected. The analytical solutions form [5] does not include the damping constant we have included in our model. This exact solution is visually not different from the simulation results with $C = 1$. The results in Fig. 24 have been computed using $C = 0.999$. To further highlight the impact of this damping, we compare the exact solution and the simulation results performed with damping constant $C = 0.999$ in Fig. 25. We also present how changing the damping constant influence the simulation results. In particular, we compare the results for $C = 0.999$ and $C = 0.997$ in Fig. 26. Additionally, we compare the results for $C = 0.999$ and $C = 0.995$ (the damping constant used in the simulation in the Baltic sea in the following section). Results are shown in Fig. 27, Where we see that using a smaller the damping constant leads to a wave which is damped out faster, as expected.

8. Baltic sea asteroid tsunami simulation

We conclude the paper with the hypothetical asteroid tsunami simulations in the Baltic sea. The initial state, modeling the asteroid hitting the water, is given by

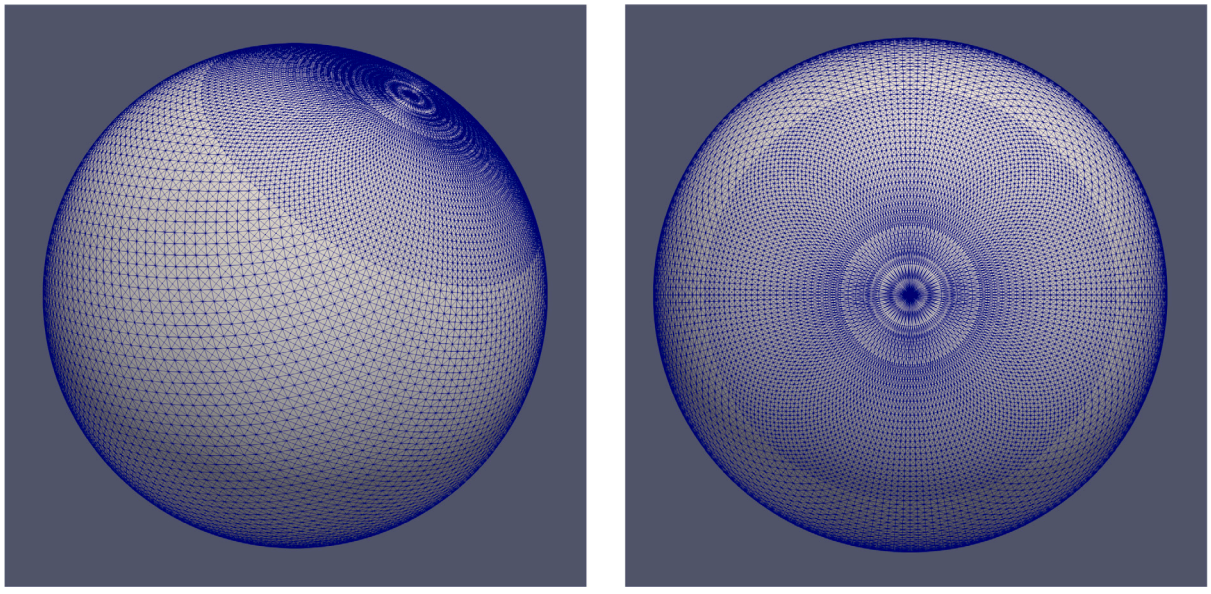
$$-A\, cos(\frac{\pi r}{n}) + A, \quad r = \left((x - x_c)^2 + (y - y_c)^2\right)^{1/2} \text{ for } r \in (-2n, 2n), \qquad (6)$$

where $A = 0.6$, $n = 47.62$, and $(x_c, y_c) = (467, 1274)$ [km] is the relative (to the corner of the mesh) location of the asteroid impact. We use



(a) Full mesh. (b) Full mesh - pole.

Fig. 11. Final mesh after 8 iterations of the longest-edge refinement algorithm where distance is calculated using x, y, z coordinates.



(a) Full mesh. (b) Full mesh - pole.

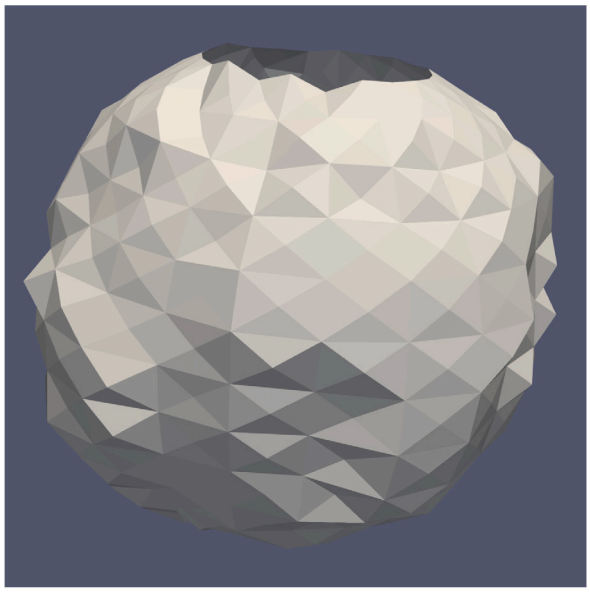
Fig. 12. Final mesh after 8 iterations of the longest-edge refinement algorithm where distance is calculated using u, v coordinates.

the non-linear wave equation (1), with damping constant $C = 0.995$, as well as the Bubnov–Galerkin finite element method and explicit time integration scheme introduced in Section 6. The damping constant represents the friction with the air and the friction with the sea bed. In our non-linear wave equations, we do not have these terms, but we represent them by the damping constant. This model parameter could be further tuned by comparing the numerical simulations of the tsunami with some available measurements of some real-life tsunami phenomena, which may be a topic of our future work. The simulation has been implemented in Julia [22] using our prototype finite element method code.

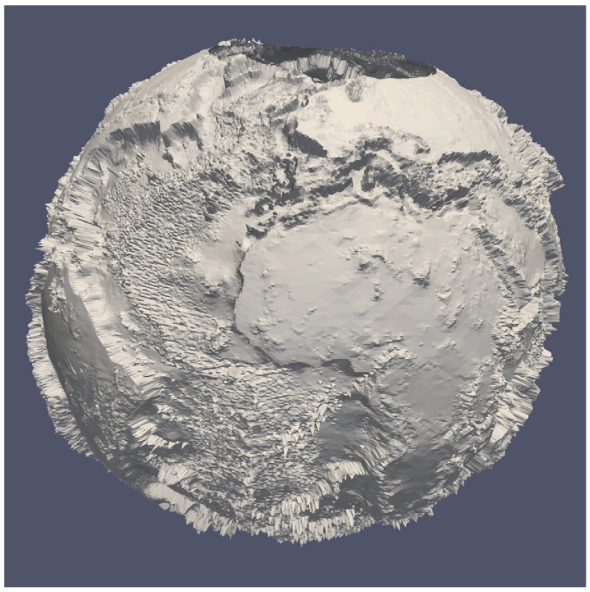
In this simulation, the part of the domain where the tsunami happens is far from the global boundary. Thus we use the same homogeneous Neumann boundary conditions, which we consider to be the acceptable assumption for the simplicity of the simulation. As is common practice, no further boundary conditions are applied on the seashore to allow the wave to flow over the terrain (land).

The simulation was performed with time step $dt = 0.01$ for 1000 steps. We have implemented our graph-grammar-based tsunami simulator in Julia [22]. We use the computational mesh generated for the Baltic sea area, as described in Section 5. First, the mass matrix for the fixed mesh is LU factorized using Julia solver [22], and then in each time step of the explicit simulation, we forward and backward substitute with the updated right-hand side.

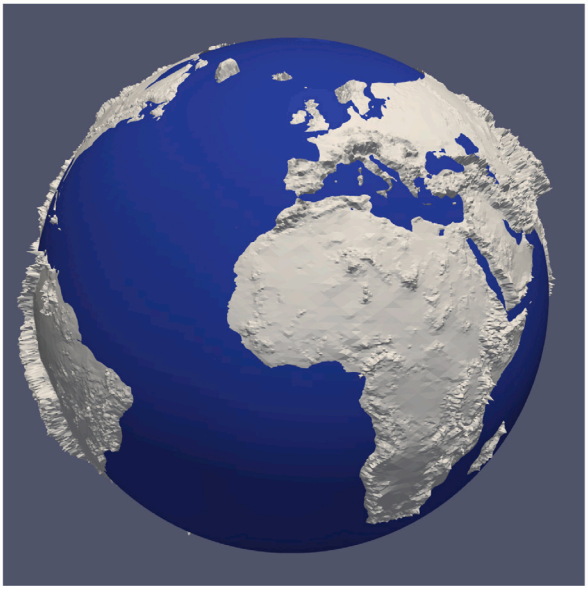
Using a laptop with an Intel Core i5-1135G7 processor, we are able to perform approximately 1000 time steps within 1 h. Snapshots from the simulation are presented in Figs. 28–32. For better visibility, the mesh (the terrain height) and the results (the sea level) in the vertical z direction are magnified 100 times. The assumed initial shape of the wave created by the asteroid hit is 1096 m above sea level; see Fig. 28.



(a) Initial mesh for terrain adaptation.



(b) Final mesh.



(c) Final mesh and the blue sphere representing water level.

Fig. 13. Mesh of the Earth with real elevation on a sphere with radius 60,000 m (compared to Earth's radius $6,371,000$ m). Distance is calculated using u, v coordinates.

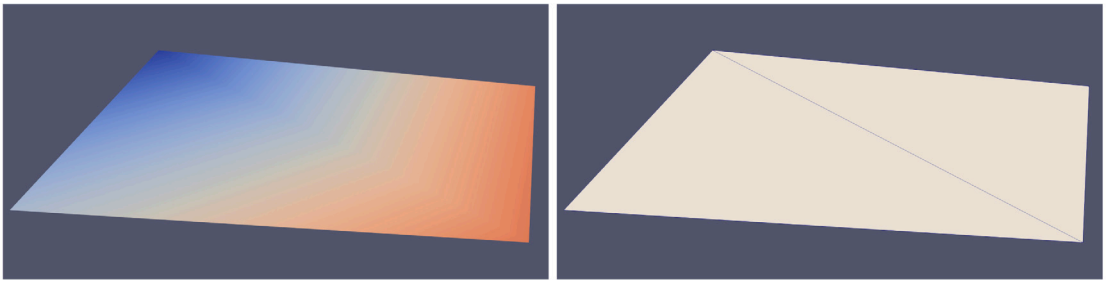


Fig. 14. Graph representing initial computational mesh.

Then, a peak in the center, up to 1616 m, is created around time step 100, as presented in Fig. 29. The tsunami wave travels in all directions, but due to the irregular structure of the seabed, the wave height is different, varying from 764 to 870 m in different directions, see Figs. 30 and 31. The wave propagates further and floods the seashore terrains, as presented in time step 1000 in Fig. 32. The tsunami waves travel

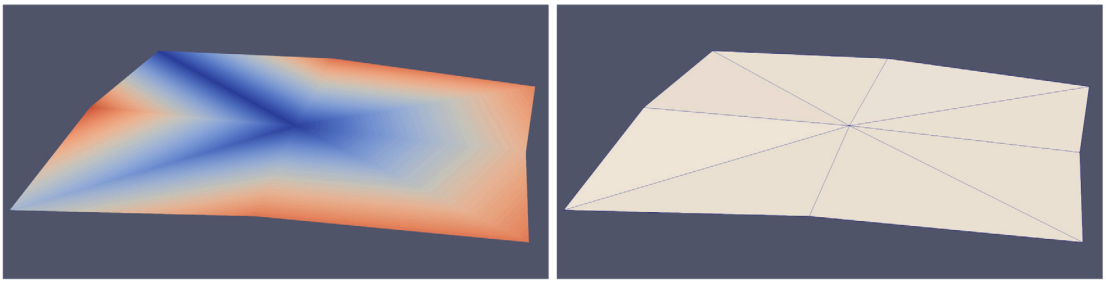


Fig. 15. Second iteration.

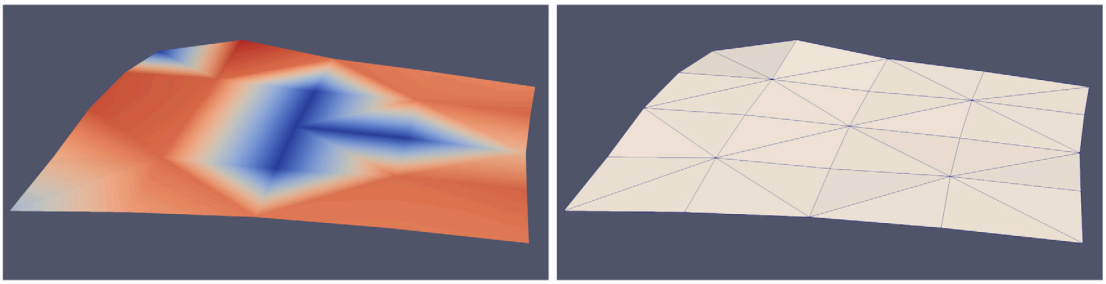


Fig. 16. Fourth iteration.

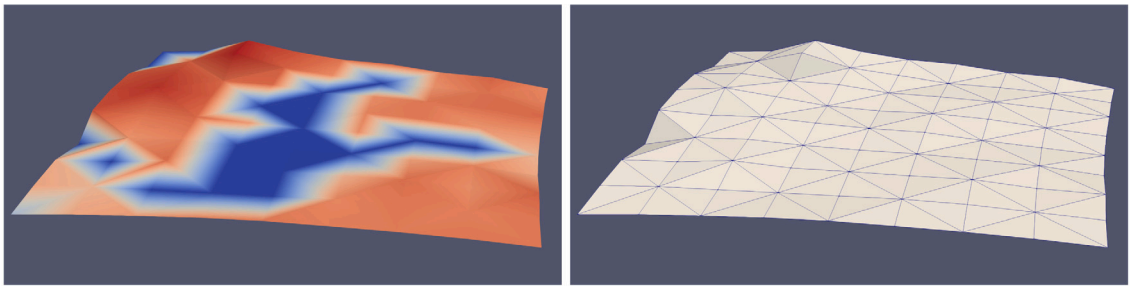


Fig. 17. Sixth iteration.

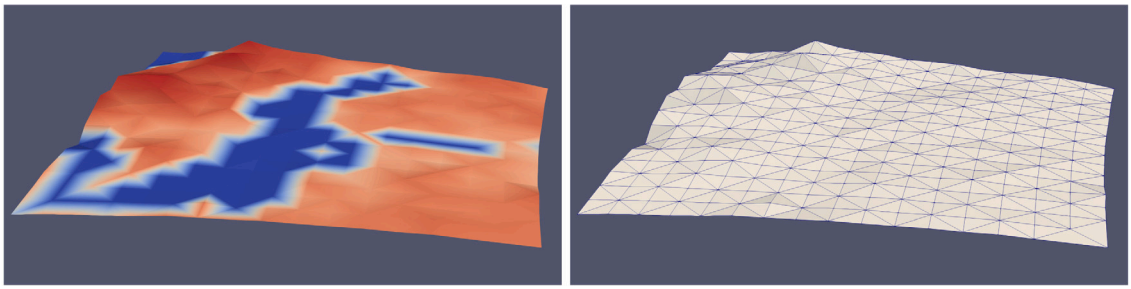


Fig. 18. Eighth iteration.

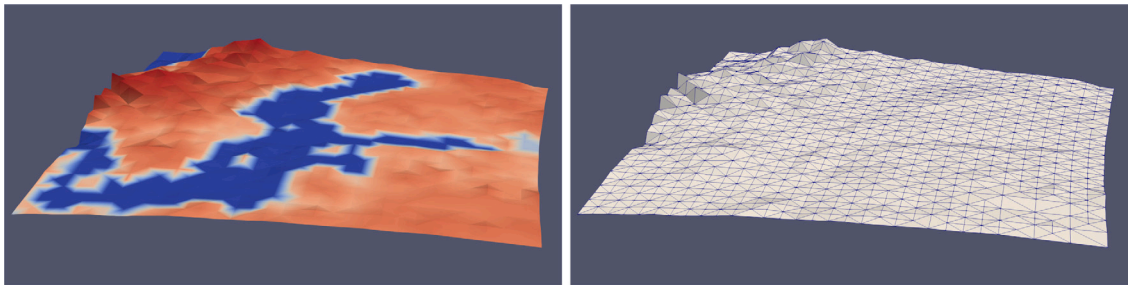


Fig. 19. Tenth iteration.

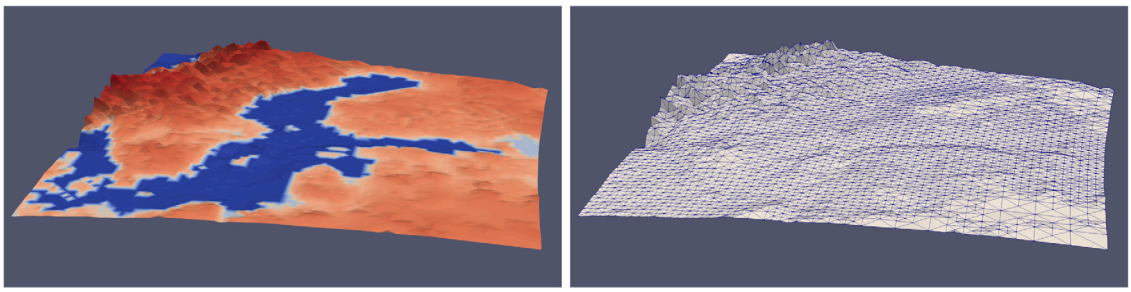


Fig. 20. Twelfth iteration.

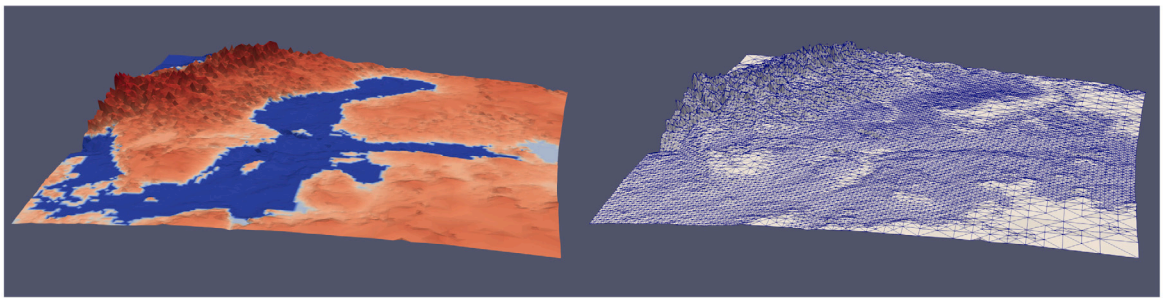


Fig. 21. Fourteenth iteration.

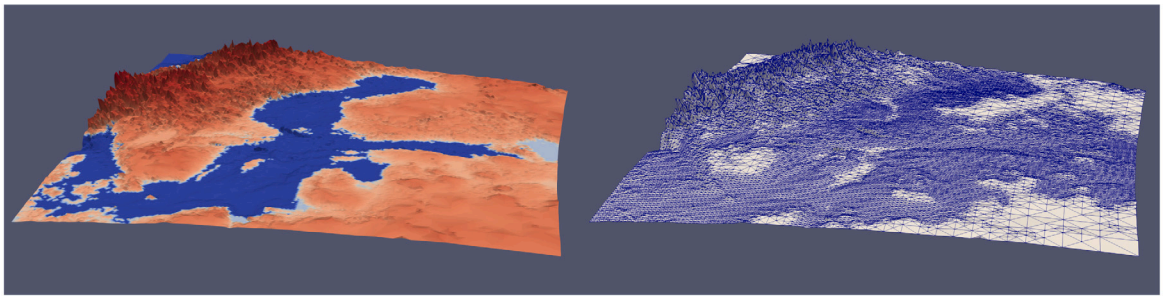


Fig. 22. Fifteenth iteration.

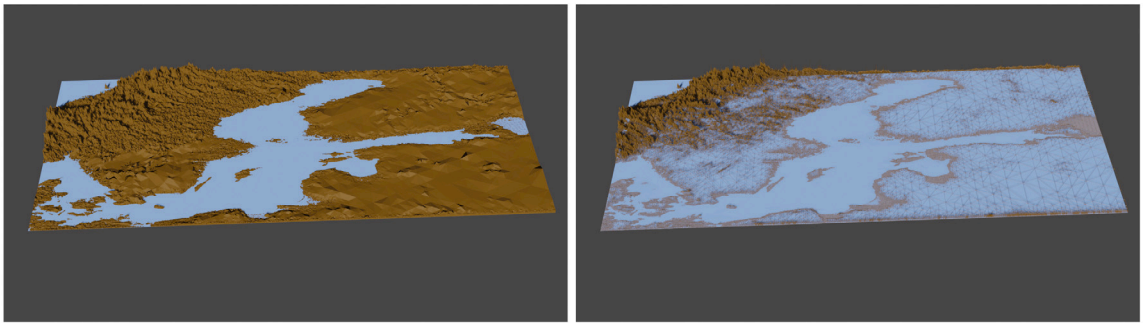


Fig. 23. The final mesh representing the Baltic sea area.

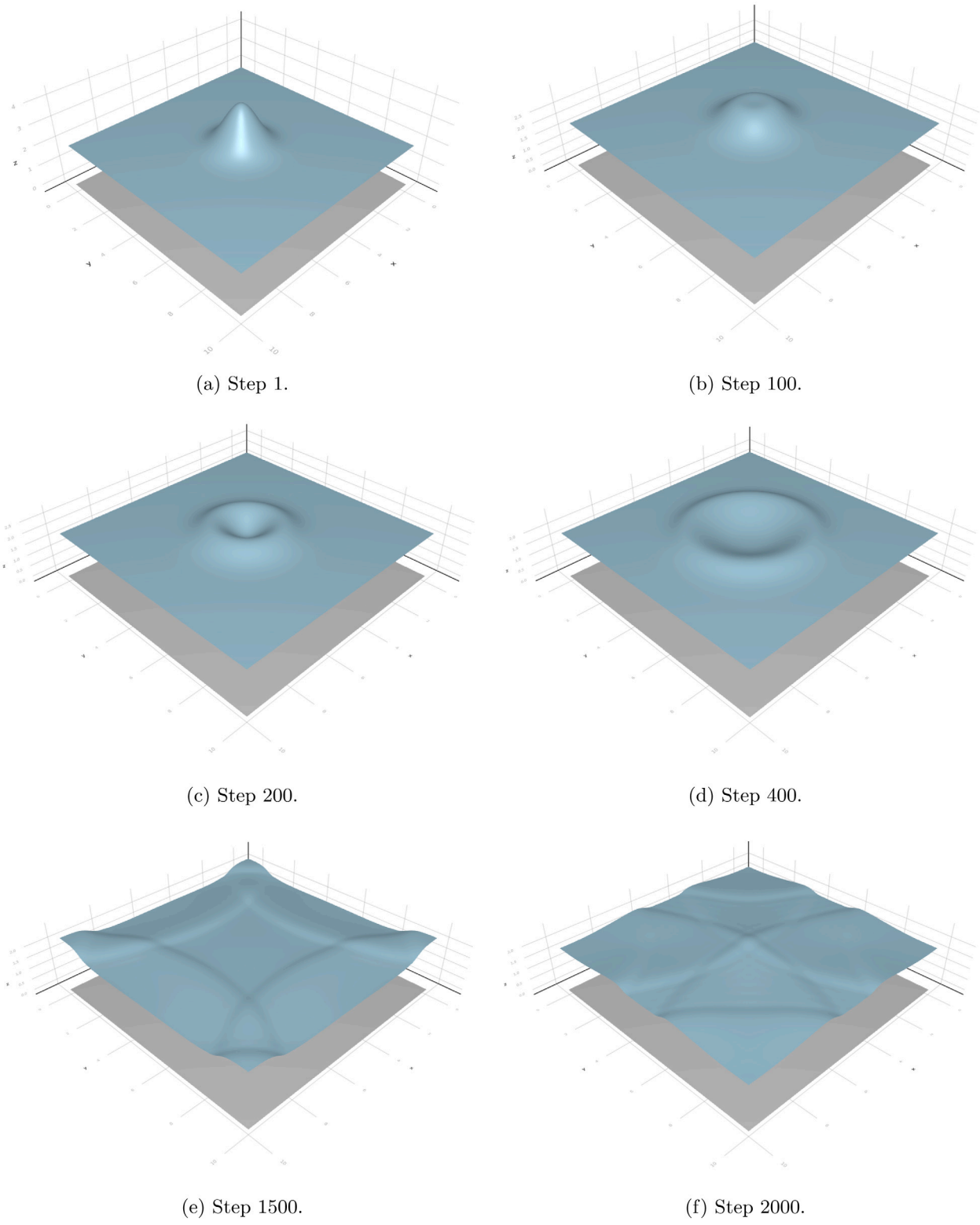
around 500 [km], and the total time of the simulation is 10,000 [s] = 2.7 [h]. This results in an average speed of propagation of 185 [km/h]. Using our graph-grammar-based platform, we can simulate and predict the impact of the local asteroid tsunami in any part of the world.

9. Computational cost of identification of the left-hand sides of graph grammar productions

In this section, we compare the computational cost of identification of the triangular element, that is, the left-hand side of a graph-grammar production, with the graph representation of the mesh obtained from

1. The six graph-grammar productions model described in [2].
2. The two graph-grammar productions model introduced in this paper.

The model employed in [2] results in a graph representation of the mesh, where element interiors are connected with vertices, and the element vertices are connected by edges. Our new model results in a graph representation of the mesh, where the element interiors are connected with edges, and edges are connected to vertices. This is illustrated in Fig. 33. We focus on the triangular element denoted by



(a) Step 1. (b) Step 100.

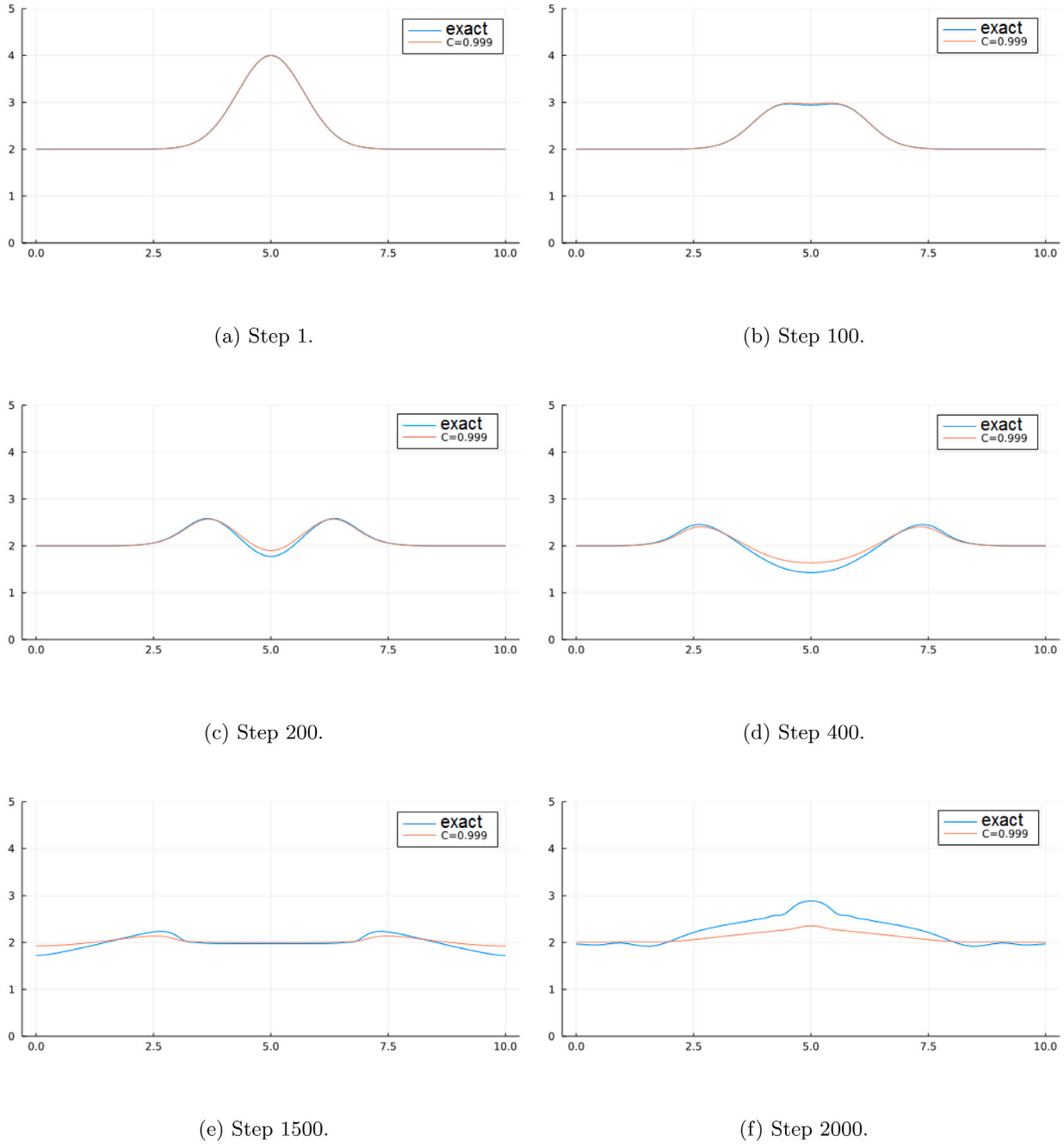
(c) Step 200. (d) Step 400.

(e) Step 1500. (f) Step 2000.

Fig. 24. Simulation of a wave in a pool filled with water.

blue. We distinguish its external boundary edge and the two internal edges, the first edge and the second edge, in the "clockwise" direction.

The number of graph node comparison operations necessary for identification of the triangle is listed in Table 1. We assume that we have a list of pointers to element interiors. During the identification of the left-hand side of the projection in the previous model, we pass from the interior node to the three vertex nodes. Now, these vertex nodes have 5, 5, or 16 adjacent graph edges. We must check the connections through the graph edges, and in the pessimistic case, to identify the common edge, we need to perform $16 \times 5 = 80$ comparisons (and



(a) Step 1. (b) Step 100.

(c) Step 200. (d) Step 400.

(e) Step 1500. (f) Step 2000.

Fig. 25. The comparison between the exact solution and the numerical simulations performed with damping constant $C = 0.999$. The simulation results for $C = 1.0$ are visually identical with the exact solution. View at the cross-section along x-axis. View along y-axis is identical.

Table 1
The number of graph node comparison operations necessary to identify the triangular elements in both representations.

Operation	Interior-vertex Model	Interior-edge Model
Identification of the first "clockwise" edge	$16 \times 5 = 80$	$4 \times 4 = 16$
Identification of the second "clockwise" edge	$16 \times 5 = 80$	$4 \times 4 = 16$
Identification of the third boundary edge	$5 \times 5 = 25$	$4 \times 4 = 16$

$4 \times 4 = 16$ for the boundary edges in this example). The upper bound on the number of operations is 80+80+25 = 185 operations.

In the new model, the interior nodes are connected with edge nodes. Now, each edge node has a maximum of 5 adjacent graph edges. In order to identify the common vertex, we must check up to $4 \times 4 = 16$ cases. The upper bound on the number of operations is 16+16+16=48 operations. In other words, the new model requires more than three times less operations in this example.

In general, an element with n_1 neighbors through the first vertex, n_2 neighbors through the second vertex, and n_3 neighbors through the third vertex, requires up to $2n_1 \times 2n_2 + 2n_1 \times 2n_3 + 2n_2 \times 2n_3$ operations to identify the triangle using the interior-vertex representation from [2], and up to $16 + 16 + 16 = 48$ operations using the interior-edge representation proposed in this paper.

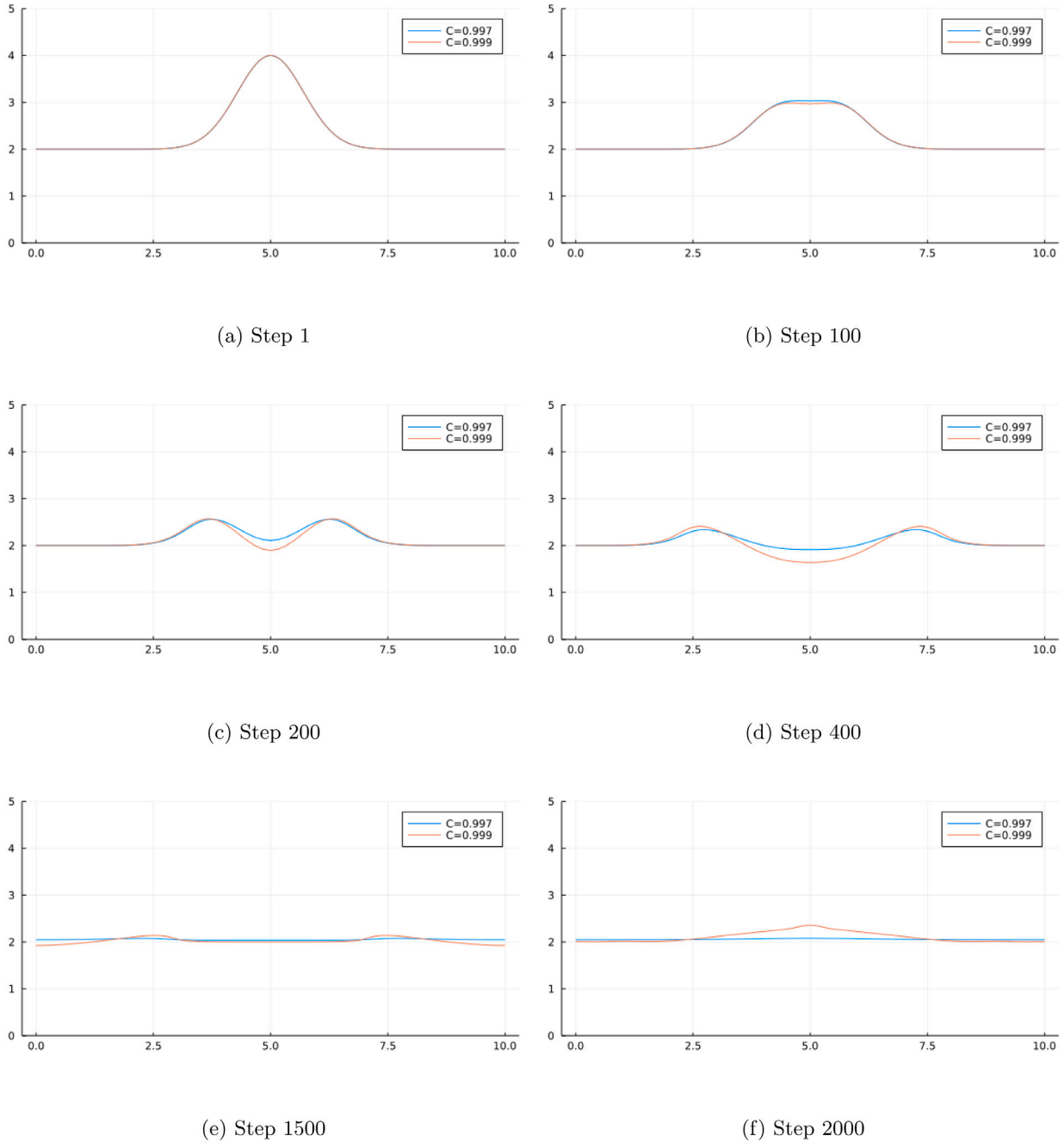


Fig. 26. Comparison of the simulations with damping constants $C = 0.999$ and $C = 0.997$. View at the cross-section along x-axis. View along y-axis is identical.

10. Conclusion

We have presented a novel graph-grammar-based platform performing simulations of asteroid tsunamis. First, we introduced the graph representation of the computational mesh where the interior of each triangle is connected with edges (not with nodes, like in our previous research). This representation reduced the computational cost of finding the subgraph isomorphic to the left-hand side graph of the production. We employed the longest-edge refinement algorithm implemented by two graph-grammar productions. The framework was implemented in Julia to perform graph-grammar-based generation of the computational mesh covering Earth, as well as detailed generation of the Baltic seabed and seashore area. Next, we implemented the non-linear hyperbolic wave propagation solver modeling the tsunami propagation phenomena. It employs LU factorization of the mass matrix and forward/backward substitutions for the consecutive time steps. We performed simulations of the tsunami caused by the asteroid falling into the Baltic sea to verify our platform. This simulation takes around 1 h on a laptop. In the simulations presented in Sections 7 and 8, we have introduced a damping parameter C. This model parameter could be further tuned by comparing the numerical simulations of the tsunami with some available measurements of some real-life tsunami phenomena, which may be a topic of our future work.

In our future work, we plan to switch to large-scale simulations of tsunami propagation on the global Earth model. We will implement our graph grammar-based solver within parallel graph transformation systems (including GALOIS [13] or Katana graph [14]. We will reformulate the non-linear wave equations (as an approximation of the

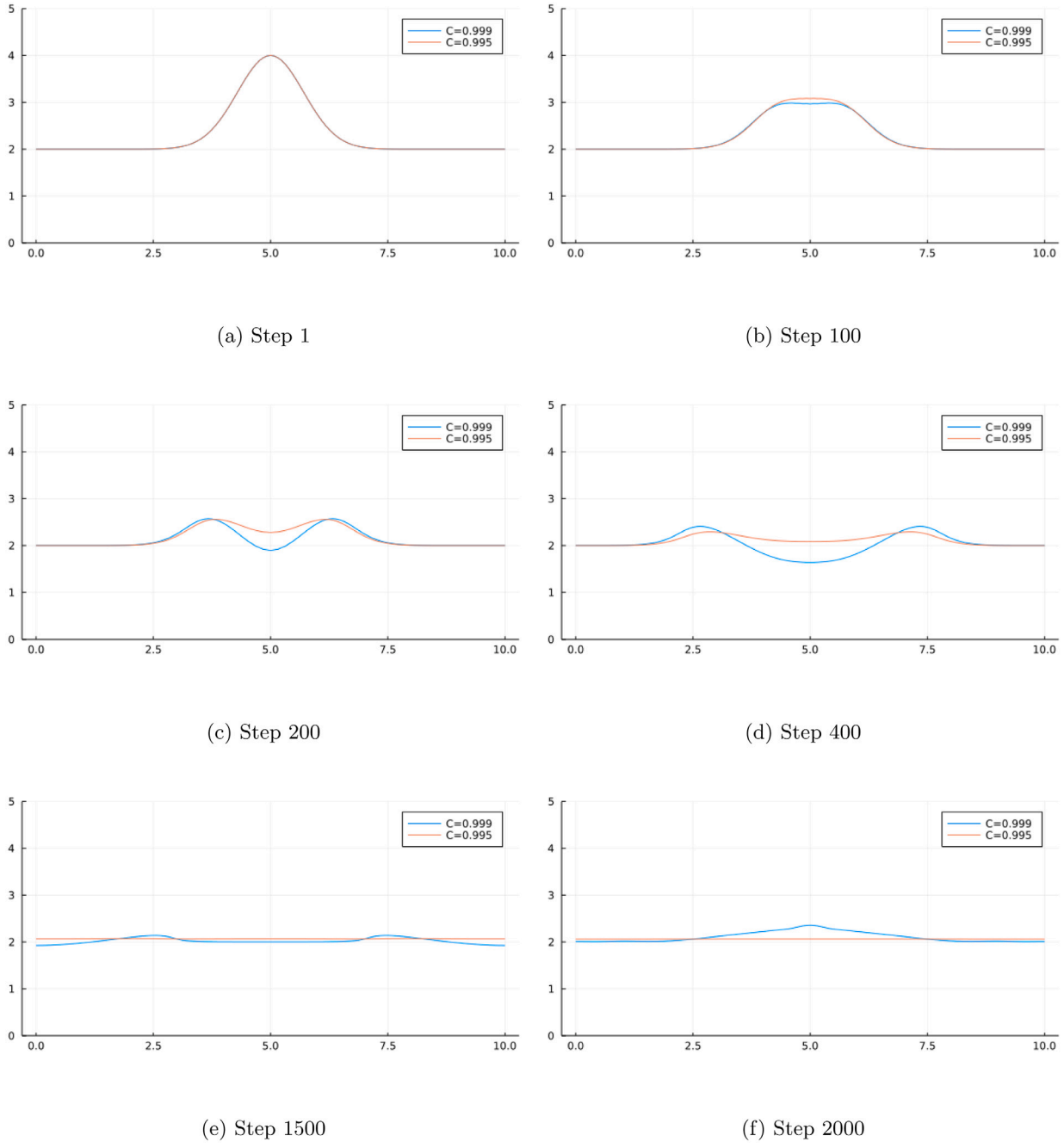


Fig. 27. Comparison of the simulations with damping constants $C = 0.999$ and $C = 0.995$. View at the cross-section along x-axis. View along y-axis is identical.

shallow water equations) in a spherical coordinate system using an implicit higher-order time integration scheme. In particular, future work will involve the development of the stabilized shallow water equation (or its non-linear wave approximation) in the spherical coordinate system, using an implicit higher-order time integration scheme, e.g., [23], allowing for large-scale simulations on the model of the global Earth. Future work will also involve parallelization into shared memory and hybrid memory Linux clusters [24,25], using the GALOIS or Katana graph environments for parallel graph transformations [13, 14]. Additionally, we plan to employ inverse algorithms [26–29] to solve several related inverse problems.

CRediT authorship contribution statement

Paweł Maczuga: Conceptualization, Formal analysis, Methodology, Investigation, Software, Validation, Visualization, Writing – original draft, Funding acquisition. **Albert Oliver-Serra:** Conceptualization, Formal analysis, Methodology. **Anna Paszyńska:** Conceptualization, Formal analysis, Methodology. **Eirik Valseth:** Conceptualization, Formal analysis, Methodology, Writing – review & editing. **Maciej Paszyński:** Conceptualization, Formal analysis, Methodology, Supervision, Writing – original draft, Writing – review & editing, Funding acquisition.

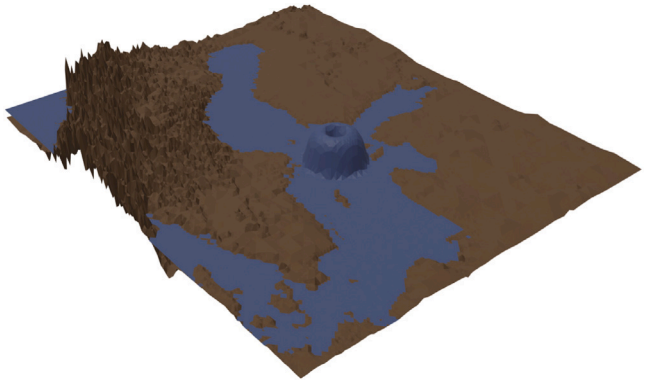


Fig. 28. Asteroid tsunami simulation time step 10.

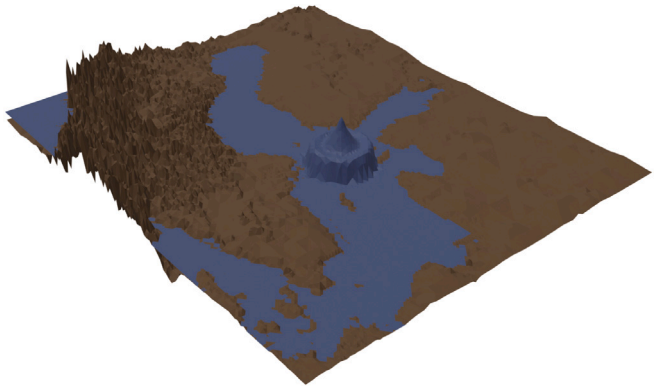


Fig. 29. Asteroid tsunami simulation time step 100.

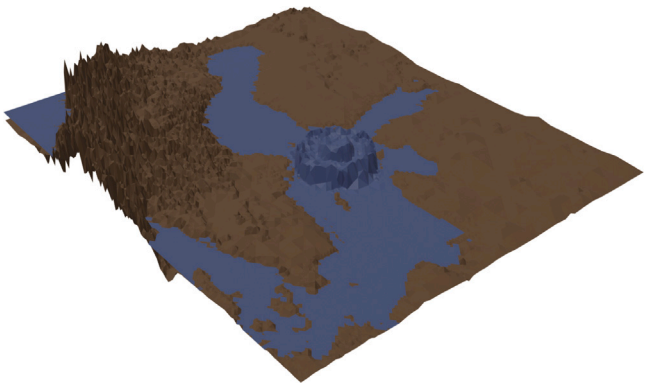


Fig. 30. Asteroid tsunami simulation time step 200.

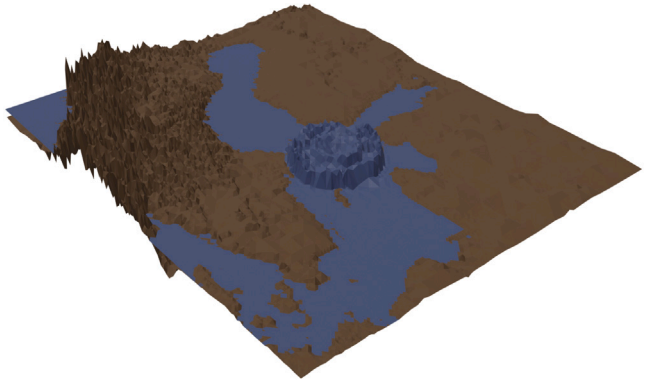


Fig. 31. Asteroid tsunami simulation time step 300.

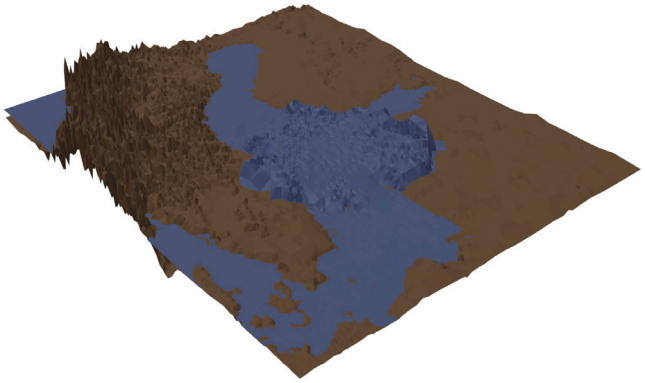


Fig. 32. Asteroid tsunami simulation time step 1000.

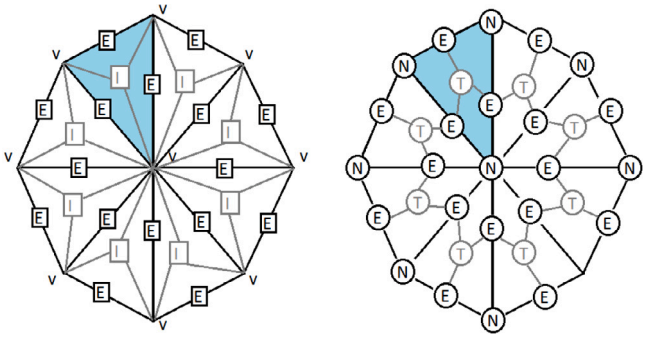


Fig. 33. Computational cost of identification of the triangular element denoted by blue color. **Left-panel:** graph representation of the mesh generated by six graph-grammar productions model. **Right panel:** graph representation of the mesh generated by two graph-grammar productions model.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The research project was supported by the program " Excellence initiative – research university" for the University of Science and Technology. The work of Eirik Valseth was supported by the Marie Skłodowska-Curie Actions grant HYDROCOUPLE - 101061623.

References

[1] https://www.nasa.gov/mission_pages/asteroids/overview/fastfacts.html.
[2] K. Podsiadło, A. Oliver Serra, A. Paszyńska, R. Montenegro-Armas, I. Henriksen, M. Paszyński, K. Pingali, Parallel graph-grammar-based algorithm for the longest-edge refinement of triangular meshes and the pollution simulations in lesser Poland area, Eng. Comput. 37 (4) (2021) 3857–3880.
[3] M.C. Rivara, Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, Internat. J. Numer. Methods Engrg. 20 (4) (1984) 745–756.
[4] M.C. Rivara, Mesh refinement processes based on the generalized bisection of simplices, SIAM J. Numer. Anal. 21 (3) (1984) 604–613.
[5] G.F. Carrier, Y. Harry, Tsunami propagation from a finite source, CMES Comput. Model. Eng. Sci. 10 (2) (2005) 113–121.
[6] A. Habel, H.J. Kreowski, May we introduce to you: Hyperedge replacement, Lecture Notes in Comput. Sci. 291 (1987) 5–26.

[7] A. Habel, H.J. Kreowski, Some structural aspects of hypergraph languages generated by hyperedge replacement, Lecture Notes in Comput. Sci. 247 (1987) 207–219.
[8] T. Heuer, P.G. Sanders, S. Schlag, Network flow-based refinement for multilevel hypergraph partitioning, J. Exp. Algorithmics 24 (1) (2019) 1–36.
[9] D.A. Papa, I.L. Markov, Hypergraph partitioning and clustering, in: T.F. Gonzalez (Ed.), Handbook of Approximation Algorithms and Metaheuristics, Chapter 61, CRC Press, Boca Raton, 2007, pp. 61–1–61–19, 2007.
[10] G. Karypis, V. Kumar, HMETIS 1.5: A Hypergraph Partitioning Package, Technical Report, Department of Computer Science, University of Minnesota, 1998.
[11] Kyle T. Mandli, Aron J. Ahmadia, Marsha Berger, Donna Calhoun, David L. George, Yiannis Hadjimichael, David I. Ketcheson, Grady I. Lemoine, Randall J. LeVeque, Clawpack: Building an open source ecosystem for solving hyperbolic PDEs, PeerJ Comput. Sci. 2, e68.
[12] Manuel J. Castro, Sergio Ortega, Carlos Parés, Well-balanced methods for the shallow water equations in spherical coordinates, Comput. & Fluids 157 (2017) 196–207.
[13] https://iss.oden.utexas.edu/?p=projects/galois.
[14] https://katanagraph.com/.
[15] Global Multi-Resolution Topography Data Synthesis, https://www.gmrt.org/.
[16] B. Romstad, C.B. Harbitz, U. Domaas, A GIS method for assessment of rock slide tsunami hazard in all Norwegian lakes and reservoirs natural hazards and earth system sciences, 2009, pp. 353–364, 9 (2).
[17] S.N. Ward, E. Asphaug, Asteroid impact tsunami: A probabilistic hazard assessment, Icarus 145 (1) (2000) 64–78.
[18] W.Y. Tan, Shallow Water Hydrodynamics: Mathematical Theory and Numerical Solution for a Two-Dimensional System of Shallow-Water Equations, Elsevier, 1992.
[19] U. Kanoglu, C.E. Synolakis, Long wave runup on piecewise linear topographies, J. Fluid Mech. 374 (1998) 1–28.
[20] Y.S. Cho, S.B. S.B. Yoon, A modified leap-frog scheme for linear shallow-water equations, Coast. Eng. J. 40 (02) (1998) 191–205.
[21] E.B. Becker, G.F. Carey, J.T. Oden, Finite Elements: An Introduction, Vol. 1, Prentice Hall, 1981.
[22] J. Bezanson, A. Edelman, S. Karpinski, V. Shah, B. Viral, Julia: A fresh approach to numerical computing, SIAM Rev. 59 (1) (2017) 65–98.
[23] Pouria Behnoudfar, Victor Calo, Marcin Łoś, Paweł Maczuga, Maciej Paszyński, A variational splitting of high-order linear multistep methods for heat transfer and advection-diffusion parabolic problems, J. Comput. Sci. (2022) in press.
[24] D. Goik, K. Jopek, M. Paszyński, A. Lenharth, D. Nguyen, K. Pingali, Graph grammar based multi-thread multi-frontal direct solver with GALOIS scheduler, Procedia Comput. Sci. 29 (29) (2014) 960–969.
[25] V. Calo, N. Collier, D. Pardo, M. Paszyński, Computational complexity and memory usage for multi-frontal direct solvers used in p finite element analysis, Procedia Comput. Sci. 4 (2011) 1854–1861.
[26] M. Paszyński, B. Barabasz, R. Schaefer, Efficient adaptive strategy for solving inverse problems, Lecture Notes in Comput. Sci. 4487 (2007) 342–354.
[27] B. Barabasz, E. Gajda-Zagórska, S. Migórski, M. Paszyński, R. Schaefer, M. Smołka, A hybrid algorithm for solving inverse problems in elasticity, Appl. Math. Comput. Sci. 24 (4) (2014) 865–886.
[28] B. Barabasz, S. Migórski, R. Schaefer, M. Paszyński, Multi-deme, twin adaptive strategy hp-HGS, Inverse Probl. Sci. Eng. 19 (1) (2011) 3–16.
[29] E. Gajda-Zagórska, R. Schaefer, M. Smołka, M. Paszyński, D. Pardo, A hybrid method for inversion of 3D DC resistivity logging measurements, Nat. Comput. 14 (3) (2015) 355–374.

Paweł Maczuga is a Ph.D. student at Institute of Computer Science, AGH University of Science and Technology, Kraków, Poland. He has got his Master Degree in 2021 under supervision of Maciej Paszyński (Title of Thesis "Parallel algorithms for the generation, adaptation and simulation of triangular meshes based on graph transformation"). His research interests include finite element method and high performance computing.

Albert Oliver Serra received his Ph.D. from the Universitat Politècnica de Catalunya and currently is an assistant professor at the University of Las Palmas de Gran Canaria. His research interest is the application of the finite element method in environmental problems. He is also working on the generation of tetrahedral adapted meshes for these problems.

Anna Paszyńska received her Ph.D. (2007) in computer science from the Institute of Fundamental Technological Research of Polish Academy of Sciences in Warsaw, Poland, and her Habilitation in Computer Science in 2019 from AGH University. She currently works as an assistant professor at the Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University in Kraków, Poland. Her research interests include graph grammars with application to finite element method simulations and direct solvers.

Eirik Valseth is a research associate in the Computational Hydraulics Group in the Oden Institute for Computational Engineering and Sciences at UT Austin. He has extensive experience in the application and development and implementation of finite element methods for challenging problems in engineering science. Since joining the Computational Hydraulics Group in 2020, Eirik has focused on the development of novel shallow water equation solvers, modeling tools and techniques for compound flooding, and the development of new ADCIRC meshes for forecasting storm surge and compound floods on the Texas Coast.

Maciej Paszyński is a full professor of Computer Science at Institute of Computer Science, AGH University, Kraków, Poland. He obtained his Ph.D. in Mathematics with Applications to Computer Science from Jagiellonian University in 2003, and his Habilitation in Computer Science in 2010 from AGH University. He co-authored over 70 papers in impact factor journals. He presented over 100 presentation at conferences and workshops. He collaborates with research groups from The University of Texas at Austin, Basque Center for Applied Mathematics in Bilbao, Spain, Catholic University of Valparaiso, Chile, Curtin University at Perth, Western Australia. His research interests include artificial intelligence and HPC for advanced simulations.