AUTOMATIC GENERATION OF LOAD-BALANCING-AWARE BLOCK-STRUCTURED GRIDS FOR COMPLEX OCEAN DOMAINS

Daniel Zint[1,2]   Roberto Grosso[1]   Vadym Aizinger[3]   Sara Faghih-Naini[1,3]
Sebastian Kuckuk[1,4]   Harald Köstler[1]

[1]*Friedrich-Alexander-University Erlangen-Nuremberg, Germany, daniel.zint@fau.de*
[2]*INRIA Sophia-Antipolis, France, daniel.zint@inria.fr*
[3]*University of Bayreuth, Germany*
[4]*Erlangen National High Performance Computing Center (NHR@FAU)*

ABSTRACT

Many high-performance computing applications involve sophisticated finite element simulations on complex domains and, for this reason, often cannot use a single structured grid for the entire domain. A popular alternative are block-structured grids (BSGs) that are more flexible geometrically but still offer a significant amount of structure. However, the standard generation process for BSGs relies heavily on manual input to define the segmentation of the computational domain – a rather difficult task to perform for complex geometries. Ocean domains often contain fractal boundary shapes and details such as islands and channels that cannot be accurately represented using BSGs. We present a method to automatically generate BSGs with an exactly specified number of blocks for real-world domains arising in 2D ocean simulations. Our BSGs consist of quad blocks refined via structured triangular grids and employ masks to accurately represent small features. The performance of the proposed BSG generation method is evaluated for realistic ocean domains and validated using simulations of the two-dimensional shallow water equations discretized by the discontinuous Galerkin method.

Keywords: mesh generation, block-structured grids, high-performance computing, discontinuous Galerkin method, ocean simulation, shallow water equations

1. INTRODUCTION

The accuracy and the computational performance of finite element models is strongly affected by the type and the quality of the employed computational mesh. Structured grids enable memory access in repeated regular stencils and therefore offer nearly optimal efficiency [1]. Discretizations utilizing unstructured meshes need to additionally load indexing data and they access memory in irregular fashion resulting in cache misses which reduce performance [2]. Nevertheless, unstructured meshes are often favored due to their geometrical flexibility – many domains with complex boundaries and varying element sizes cannot be accurately represented by structured grids at all – and the ability to adapt resolution in accordance with the application requirements. Furthermore, the grid structure also plays an important role for load balance in distributed computations. As pointed out in [3], the grid should be adapted to the hardware that is used for the simulation. In addition, the size of mesh elements determines the maximum admissible time step in explicit simulations of time-dependent problems, whereas the element shape critically affects the stability of a finite element discretization. A triangular element, for example, is considered optimal if it is equilateral; the more it is distorted the worse its quality. Element quality can be measured by the mean ratio

metric [4, 5],

$$q_{mrm} = 4\sqrt{3} \frac{A}{\sum_{i=0}^{3} l_i^2}, \qquad (1)$$

where A is the signed area of the triangle (the sign indicates flipped triangles), and l_i are the lengths of its edges. Numerical errors caused by low-quality elements degrade the results or may even cause a blow-up of the simulation. Also, the element size has an influence on the discretization error.

A compromise between the performance of a structured grid and the flexibility of an unstructured one is a block-structured grid (BSG). It consists of an unstructured block-mesh where each block contains a structured grid. BSGs certainly alleviate the problem with complex domains but do not solve it completely. Since BSGs are complicated to generate automatically they are mostly used for simple domains, and the block structure is usually optimized for specific applications (e.g. turbine blades). For ocean domains, no such simple segmentation is possible; in addition, real-world geometries often contain application-critical small-scale features. This presents a major difficulty for the BSG methodology: Given a certain minimum block size, islands or other domain features smaller than this size cannot be represented.

In the current work, we introduce and evaluate a masking approach aiming to solve this issue: Our BSGs are generated for simplified geometries that do not resolve features smaller than the given block size; instead, the excessive elements (those outside of the correctly resolved geometry) are masked, i.e. excluded from the simulation. Starting from a user-provided unstructured triangular mesh, our method automatically generates a BSG of a given density with a prescribed number of topologically uniform blocks. Optimal load balance in a parallel simulation is achieved by choosing the number of blocks to be a multiple of the processing units. Complex boundaries and small islands that usually cannot be represented with BSGs are restored by masking elements and repositioning boundary vertices. The code is available at `https://github.com/DanielZint/hpmeshgen`.

This paper is structured as follows. In Section 2, we describe related work. The generation of the block structure is presented in Section 3. The refinement of blocks, the masking, and the adaptation to the domain are described in Section 4. BSGs for selected real-world domains and simulation results used for validation of our approach can be found in Section 5, and a short Conclusions & Outlook section wraps up this work.

2. RELATED WORK

Considering that BSGs are used in many high-performance computing applications, e.g. [6, 7, 8, 9], there is surprisingly little literature on generating such grids. Armstrong et al. showed in [10] that methods for generating BSGs share the same difficulty, namely the placement of mesh singularities. Fogg et al. use cross-fields for generating a block structure [11, 12]. Lim et al. propose an evolutionary algorithm for block generation [13, 14]. Sánchez and Cruz present a semi-automatic approach for parametrizing polygonal regions [15], in which a polygonal region is decomposed into quadrilateral blocks and refined via structured quad grids. By enabling manual correction of the decomposition, the blocks are large and results look promising. A similar approach was presented in [16]. However, these methods focus on rather simple domains which are decomposed into a small number of blocks. The ocean domains in the focus of the current study are much more complex geometrically, and we have additional constraints such as the exactly prescribed number of blocks and the CFL condition, Equation (2).

A method for generating BSGs for complex ocean domains was presented in [17]. It takes into account the CFL condition, but its performance is limited by the domain geometry: Realistic coastal regions with fractal shapes and small islands cannot be represented accurately. In addition, the method in [17] does not produce an exact number of blocks and therefore may cause load imbalances. Nevertheless, our current scheme follows the same idea of generating blocks by simplifying an unstructured triangular mesh.

BSG generation also appears in geometry processing where blocks are used for efficiently storing textures and the grid itself. Boier-Martin et al. [18] and Carr et al. [19] use clustering techniques for block creation. Dong et al. [20] quadrangulate any manifold by applying a Morse-theoretic analysis to the eigenvectors of the mesh Laplacian. Daniels et al. present an algorithm for quadrilateral remeshing [21]. It requires closed manifold meshes and is therefore not transferable to 2D ocean meshes. None of the above methods considers element quality as the meshes are not designed for numerical simulations.

Campen [22] presented a survey of methods for partitioning surfaces into quadrilateral patches. The methods presented there have a different objective. Blocks do not have a prescribed size, and the number of blocks is also not fixed. Therefore, these methods are not appropriate for HPC.

3. GENERATION OF BLOCK STRUCTURE

To generate a quad block structure with a prescribed number of blocks, we first simplify the unstructured triangular mesh, Figure 1a, to twice the prescribed number of blocks, Figure 1b. The coarse triangles are then merged into quads to form a quad block structure, Figure 1c. The quad blocks are refined with structured triangular grids, Figure 1d. Elements that are outside the domain are masked, Figure 1e. Finally, boundary vertices are mapped to the original contour, and element size is restored, Figure 1f.

Quad blocks with structured triangular grids offer advantages and disadvantages: On the one hand, one needs only half as many quad blocks as triangular ones, and the communication topology between quad blocks is simpler to optimize; on the other, it is easier to produce an accurate representation of complex boundaries by masking triangular grids. In addition, the generation process for triangular meshes (used for partitioning into blocks) is robust and produces high-quality partitions, whereas robustly generating unstructured partitions into quads without degenerated elements is a much more complex task. A triangular partition can be converted into a quad one by combining triangles [23, 24]. The resulting quad mesh might have degenerated quads, e.g. the two quads in the top of Figure 1c, but the triangles inside the quad blocks are still valid, Figure 1d. Furthermore, our grid generator was developed for the ExaStencils [25] code generation framework with its python front-end GHODDESS [26] currently limited to quad-type communication topologies.

Aside from complex boundary regions, also the element size must be considered in the mesh generation process. The CFL condition for shallow water equations contains a quotient that describes the relation between element size Δx and ocean depth H [27],

$$c_m = \frac{\Delta x}{\sqrt{H}}. \tag{2}$$

The largest possible time step is proportional to c_m,

$$\Delta t_{\max} \sim c_m = \frac{\Delta x}{\sqrt{H}}. \tag{3}$$

Large elements allow large time steps and therefore faster computation, but the simulation also becomes less accurate. Thus, a compromise has to be found between time step size and accuracy. The element with the smallest c_m determines the maximum time step $\Delta t_{\max}$, whereas the discretization error (and thus the accuracy) is largely controlled by Δx. Therefore, c_m should be approximately the same for each element, whereas the local mesh resolution should be chosen to provide sufficient numerical accuracy.

3.1 Simplification

Our method for simplifying the triangular mesh is based on the ideas of [17]. However, we use a different error metric and vertex positioning method.

Quadric mesh simplification modifies a triangular mesh by performing edge collapses using the quadric error metric [28]. First, the error of all edge collapses is computed. Simplification is then performed iteratively starting with the collapse that causes the smallest error. After each edge collapse, the error of the surrounding edges is recomputed. The simplification terminates when the desired number of triangles is reached or when no more edges can be collapsed.

Error Metric

Definition 1 *The* relative distance $\tilde{d}(\mathbf{p}_i, \mathbf{p}_j)$ *between two points* $\mathbf{p}_i$ *and* $\mathbf{p}_j$ *is the quotient of the Euclidean distance* $\|\mathbf{p}_j - \mathbf{p}_i\|$ *and the integral of the size function* $h(\mathbf{x})$ *between the two points divided by the Euclidean distance:*

$$\tilde{d}(\mathbf{p}_i, \mathbf{p}_j) = \frac{\|\mathbf{p}_j - \mathbf{p}_i\|}{\frac{\int_{\mathbf{p}_i}^{\mathbf{p}_j} h(\mathbf{x})d\mathbf{x}}{\|\mathbf{p}_j - \mathbf{p}_i\|}} = \frac{\|\mathbf{p}_j - \mathbf{p}_i\|^2}{\int_{\mathbf{p}_i}^{\mathbf{p}_j} h(\mathbf{x})d\mathbf{x}}. \tag{4}$$

A detailed explanation on generating the size function $h(\mathbf{x})$ from a triangular mesh is given in [17].

Definition 2 *The* relative length r_e *of an edge* e *is the relative distance of its incident vertices* v_i *and* v_j *with positions* $\mathbf{p}_i$ *and* $\mathbf{p}_j$,

$$r_e = \tilde{d}(\mathbf{p}_i, \mathbf{p}_j). \tag{5}$$

Definition 3 *The* position error $\rho_v(\mathbf{x})$ *of a vertex* v *is the squared relative distance between its initial position* $\mathbf{p}$ *and its current position* $\mathbf{x}$,

$$\rho_v(\mathbf{x}) = \tilde{d}(\mathbf{p}, \mathbf{x})^2. \tag{6}$$

We derive the simplification error from the position error $\rho_v(\mathbf{x})$. To ensure that the mesh is simplified uniformly we keep track of all vertices that were collapsed into a single vertex v by storing them in a set $V_c(v)$. The simplification error $Q_v(\mathbf{x})$ of vertex v at position $\mathbf{x}$ is the sum of all position errors of the vertices that were collapsed into v

$$Q_v(\mathbf{x}) = \sum_{v_c \in V_c(v)} \rho_{v_c}(\mathbf{x}). \tag{7}$$

The error of an edge collapse is the sum of the simplification errors of its vertices v_i and v_j,

$$Q_e = Q_{v_i} + Q_{v_j}. \tag{8}$$

(a) initial mesh   (b) simplification   (c) blocks   (d) BSG topology   (e) masking   (f) optimized BSG
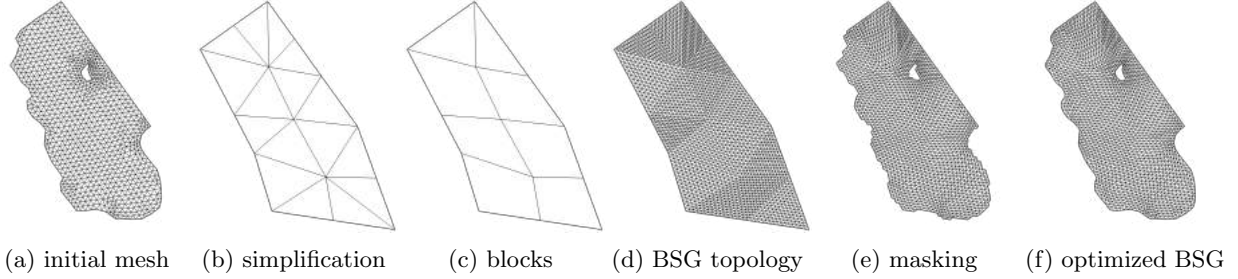
Figure 1: The BSG generation steps for the Bahamas domain.

When collapsing v_j into v_i, the set $V_c(v_j)$ is appended to $V_c(v_i)$.

Several conditions have to be met for an edge before it can be collapsed. An edge collapse is considered *invalid* if

- the collapsing edge connects two boundary vertices but itself is in the interior, or

- the resulting elements have poor quality.

The first condition ensures that the mesh is not cut open. The second condition prohibits flipped and deformed triangles. We consider a triangle as low quality if the mean ratio metric, Equation (1), is below 0.1. Similar constraints, called link conditions, were formulated by Dey et al. [29].

Once an interior boundary (usually an island) only contains three edges, it is replaced by a triangle. Thus, islands disappear if they are too small to be represented in the block structure. Additionally, we make use of non-edge contractions introduced in [30]. Vertices that are close but not connected by an edge are collapsed along a virtual edge.

Vertex Positioning

The computation of vertex positions after an edge collapse differs for interior and boundary vertices. If both vertices of the collapsed edge are in the interior or the edge is virtual, the remaining vertex is positioned in the middle. More elaborate approaches like optimizing the new vertex position were tested but did not have any significant impact on quality. If an edge connects an interior with a boundary vertex, the edge is collapsed towards the boundary. This ensures that the domain shape remains unchanged.

For boundary edges, we consider several cases depending on the local convexity of the boundary at the two vertices. If the boundary is concave at both vertices, the new vertex is positioned at the midpoint of the

edge, Figure 2a. If both vertices are convex we compute the intersection point of the neighboring edges and use it as the new vertex position, Figure 2b. If one vertex is convex and the other is concave, the convex vertex position is preserved, Figure 2c. This only leads to a valid solution if these edges are not parallel. Otherwise, the edge collapse is considered invalid. The positioning of boundary vertices ensures that the initial mesh is fully covered by the simplified mesh.
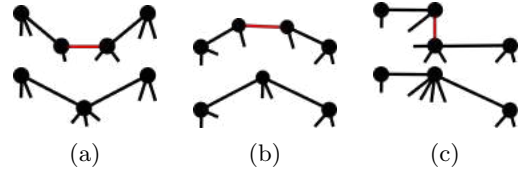

(a)   (b)   (c)

Figure 2: Vertex positioning at boundaries. The collapsed edge is marked in red.

3.2 Remeshing

The simplified mesh is improved with standard postprocessing steps like smoothing and edge flipping. We perform remeshing similarly to [31]. First, we compute the average of the relative edge lengths $\bar{r}_e$. Then, the mesh is iteratively improved with the following steps:

1. Split all edges whose relative length is larger than $\frac{4}{3}\bar{r}_e$.

2. Collapse all edges whose relative length is smaller than $\frac{3}{4}\bar{r}_e$. Collapses are executed as described in Section 3.1.

3. Flip edges whenever it reduces the number of irregular vertices. A vertex is considered irregular if it has a valence unequal to 6. An edge with the vertices v_1, v_2 and the incident triangles

28

(v_1, v_2, v_3), (v_2, v_1, v_4) is flipped if $e'_\eta < e_\eta$, where

$$e_\eta = \max\{|\eta_1 - 6|, |\eta_2 - 6|\}$$
$$+ \max\{|\eta_3 - 6|, |\eta_4 - 6|\},$$
$$e'_\eta = \max\{|\eta_1 - 7|, |\eta_2 - 7|\}$$
$$+ \max\{|\eta_3 - 5|, |\eta_4 - 5|\},$$

and η_i is the *valence* of vertex v_i (i.e. the number of vertices connected to v_i by an edge). The valence of a boundary vertex v_b is increased depending on the boundary angle α_b,

$$\eta_b \leftarrow \eta_b + \texttt{floor}(\frac{2\pi - \alpha_b}{\frac{1}{3}\pi}).$$

Flips that cause tangling, i.e. triangles with negative quality, Equation (1), are discarded.

4. Improve mesh quality by smoothing. We use the DMO (Discrete Mesh Optimization) approach from [32] with a vertex metric that combines the mean ratio metric, Equation (1), with Laplace smoothing

$$q_{iso}^{(v)}(\mathbf{x}_k) = \begin{cases} q_{mrm}^{(v)}(\mathbf{x}_k), & \text{if } q_{mrm}^{(v)}(\mathbf{x}_k) < 0.5 \\ 0.5 + q_{lap}^{(v)}(\mathbf{x}_k), & \text{otherwise} \end{cases}$$
$$q_{lap}^{(v)}(\mathbf{x}_k) = \frac{1}{\|\mathbf{lp}_k - \mathbf{x}_k\|^2 + 1},$$

where $\mathbf{x}_k$ is the position of vertex v_k, and $\mathbf{lp}_k$ denotes the center of gravity of the one-ring neighborhood (vertices connected to v_k by an edge). The advantage of this metric over the pure mean ratio is that the results are smoother, and elements tend to be locally of similar size. The mean ratio metric can cause distortions if mesh topology is not adequate for the domain.

Remeshing is stopped if either the number of triangles does not change within two iterations or if a certain maximum number of iterations is reached (currently, we perform a maximum of 100 iterations). After that, the number of triangles is not generally equal to the number of blocks as prescribed by the user; thus, the last remeshing iteration is carried out to produce the exact number of blocks: We force edge splits if we do not have enough triangles or we collapse them if there are too many – independently of $\bar{r}_e$.

An example showing the effectiveness of remeshing is given in Figure 3. Low quality triangles and vertices with high valence are removed by remeshing, improving the overall mesh quality. Both, the simplified and the remeshed mesh consist of 4000 triangles.

3.3 Conversion to Quad Blocks

The triangles in the simplified mesh are merged using Blossom-Quad [23] which relies on Edmonds' Algorithm to find a perfect match for the dual graph of
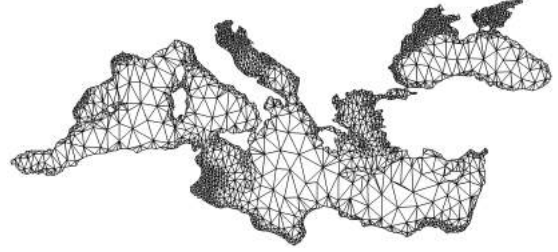
(a) Simplified
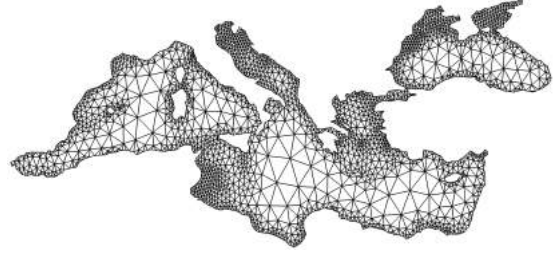
(b) Remeshed

Figure 3: Remeshing for domain Mediterranean with 4000 triangles.

the triangular mesh. Unfortunately, not every graph has a perfect match and therefore some triangles might be left over. These triangles always appear pair-wise along mesh boundaries. The solution proposed in [23] duplicates the vertex between the two triangles. This solution is not feasible for us because this increases the number of quads. Instead, we perform triangle merging as proposed several times in literature [24, 33, 34]. This method moves one triangle towards another by flipping edges until they can be merged into a quad.

The mesh must have an even number of triangles for applying Blossom-Quad and triangle merging. The simplification and remeshing might cause an uneven number though. In that case, the relatively longest boundary edge according to Equation (5) is split in two increasing the total number of triangles by one.

Finally, the quad blocks are refined with structured triangle meshes, giving the desired block-structured topology. There are two possible orientations for a triangle mesh within a quad block. We choose the orientation which results in triangles of higher quality, measured with the mean ratio metric, Equation (1).

4. GRID ADAPTATION TO DOMAIN

The grid adaptation consists of three main steps. First, element size is adjusted by repositioning interior vertices. Second, the domain shape is restored by masking triangles. Third, boundary vertices are mapped onto the domain shape. Finally, interior vertices are repositioned once more to improve quality

near boundaries.

4.1 Repositioning Interior Vertices

For finding optimal vertex positions in the interior of the BSG, we define a quality metric and optimize vertex positions with DMO [32]. We use the relative edge length defined in Section 3.1 to adapt mesh density. Additionally, we set a minimal mean ratio quality $\hat{q}_{mrm}^{(v)}$ to ensure numerical stability:

$$q_{d,mrm}^{(v)}(\mathbf{x}) = \begin{cases} q_{mrm}^{(v)}(\mathbf{x}) & \text{if } q_{mrm}^{(v)}(\mathbf{x}) \leq \hat{q}_{mrm}^{(v)} \\ \hat{q}_{mrm}^{(v)} + q_d^{(v)}(\mathbf{x}) & \text{otherwise.} \end{cases}$$

The density quality $q_d^{(v)}(\mathbf{x})$ of vertex v at position $\mathbf{x}$ depends on the longest and shortest relative length of its incident edges:

$$q_d^{(v)}(\mathbf{x}) = \frac{1}{r_{e,max} - r_{e,min} + 1}.$$

Optimizing for density quality sometimes generates wiggly lines. We remove them by applying one iteration of DMO with the mean ratio metric.

4.2 Masking

We trim the mesh such that it represents the domain well by masking elements that are outside of the domain. For this, a signed distance function is utilized to decide which vertices and edges give the best representation of the boundaries. In the first sweep, we mask all triangles that lie outside the domain. This is determined by checking the signed distance of all incident vertices and the center point of the triangle. If all distances are positive, the triangle is masked. Considering the center point of the triangle prevents masking those triangles that have all vertices on the boundary but should not be masked. Due to round-off effects, boundary vertices may have positive signed distances even though they lie on the contour.

In contrast to triangles that must be preserved, it might also happen that triangles lie almost completely outside the domain. For masking those triangles we approximate the area fraction of the triangle that is outside of the domain. If its major part (currently we use a threshold of 85 %) lies outside, it is masked.

Next, we consider boundary vertices. We mask a vertex and its incident triangles if the boundary is approximated better by the vertices on its one-ring. For that we compare the distance of the vertex with all its neighbors. If all neighbors are closer to the contour than the vertex, it is masked.

One more special case must be accounted for before mapping the boundary to the contour. Trimming might cause a zigzag line at the boundary. If this line

is mapped to the contour, triangles might degenerate. We avoid this by masking triangles that would be of low quality when mapped to the contour.

4.3 Boundary Adaptation

Producing non-degenerate triangles is more important than placing boundary vertices perfectly on the domain boundary. Thus, vertices are only mapped onto the contour if the resulting triangles are not degenerated. Fortunately, this happens only rarely. Vertices are mapped by moving them to the closest point on the contour. Afterwards, all boundary vertices are smoothed by positioning them in the midpoint between their neighbors.

5. RESULTS

In this section, we compare BSGs to the initial unstructured triangular meshes. Element quality is measured with the mean ratio metric, Equation 1, element size is evaluated by the CFL quotient, Equation 2. For the BSG generation, we use a workstation with an Intel i7-6700k CPU with 4 cores and 4.0 GHz and an NVIDIA GeForce GTX 1070 GPU. Vertex repositioning is performed on GPU everything else on CPU. All runtime measurements cover the whole generation process including read and write operations.

The number of triangles $|T|$ in a BSG is the product of the number of blocks N_B and the number of elements per block U. The structured triangular grid is generated by refining each block uniformly. The number of unmasked triangles is denoted by $|F|$ and the relative amount of masked triangles by μ

$$\mu = (|T| - |F|)/|T|. \tag{9}$$

5.1 BSG generation for real-world ocean domains

The first example is called Graysharbor, Figure 4a, and represents the geometry and topography of the Grays Harbor in the State of Washington (USA). The element size varies strongly throughout the domain. We generate a BSG with $N_B = 250$, $U = 128$, and $|F| = 27\,898$. The generation of the BSG took about 15 seconds. The BSG for Graysharbor contains 13% masked triangles. The minimal mean ratio quality is 0.32. The CFL quotient varies between 83 and 4262. It is similar to the range of the unstructured mesh that is between 81 and 5547. The domain is represented correctly by the BSG, Figure 4b.

The second example is an unstructured mesh representing the Gulf of Mexico with $14\,269$ triangles, Figure 5a. The generation of a BSG with $N_B = 300$, $U = 128$, and $|F| = 32\,635$ took 8 seconds, Figure 5b.
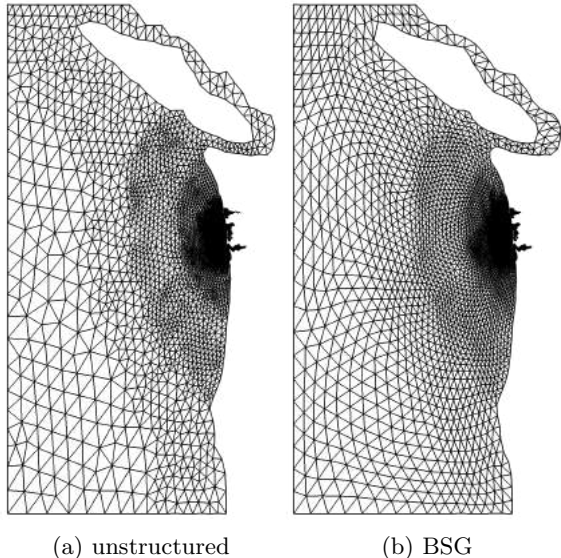
Figure 4: Graysharbor with 34 406 triangles in the unstructured mesh [35] and 27 898 in the BSG.

The minimal mean ratio quality for the BSG is 0.30. Although the BSG has more than double the number of elements, the CFL quotient is very similar for both grids. The unstructured mesh has a CFL quotient between 36 and 11580 and the BSG between 34 and 7304. In this BSG, 15% of the elements were masked.
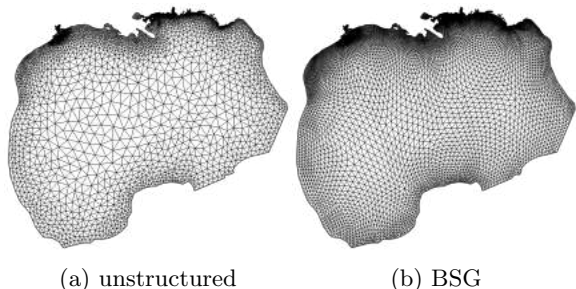

Figure 5: The Gulf of Mexico with 14 269 triangles in the unstructured mesh and 32 635 in the BSG.

5.2 Validation test: Mediterranean

The domain Mediterranean is our most complex example; it contains fine-scale geometry features such as small islands and channels which are only one element wide, Figure 6. The unstructured mesh consists of 112 962 triangles, and we generate BSGs with up to 8 000 blocks and a higher resolution than the initial unstructured mesh in order to represent details like the Bosporus connecting the Black Sea with the Marmara Sea correctly. Due to the increased number of elements, the CFL quotient is smaller for BSGs than for the unstructured mesh. For $N_B = 4000$ with 472 931 triangles, we have a minimal value of $c_m = 35.1$. The mean ratio metric has its minimal value at around 0.3 for all meshes. BSG generation for the Mediterranean domain takes between 103 and 141 seconds.

For this domain, we clearly see the advantage of masking elements. Even small details can be captured by BSGs with masks. In Figure 7 we illustrate some details of the BSG with $N_B = 8\,000$. Regions such as the Bosporus, where the mesh must be connected, are represented correctly, Figure 7a. In other regions such as the Gulf of Corinth, the isthmus is also shown correctly in the BSG, Figure 7d. The small land bridge connecting southern Greece with the Peloponnese cannot be represented on the block level but is restored by masking in the refined mesh. Sicily, on the other hand, is correctly shown as disconnected from the mainland of Italy, Figure 7h. Coastal regions are represented substantially better by the refined and masked grid than by the unmasked block grid, Figure 7f. Most of these details could not be represented without masking as triangles would be highly distorted, e.g imagine fitting whole blocks into the Bosporus, Figure 7a.

N_B	U	$|F|$	μ	runtime/s
8 000	32	244 295	5 %	103
4 000	128	472 931	8 %	141
2 000	128	226 350	12 %	137

Table 1: BSG configurations for Mediterranean with masks, where N_B, U, and $|F|$ are the number of blocks, elements per block, and unmasked elements respectively. μ is the relative amount of masked elements.

To validate the quality of the generated BSGs we additionally simulate a circulation scenario for the Mediterranean and compare results between the initial unstructured mesh and various generated BSGs. The simulations are performed using the 2D shallow water equations solver UTBEST [27] based on the discontinuous Galerkin (DG) method. The tide-driven flow is simulated for 5 days starting from the cold start conditions (zero elevation and velocity) and uses piecewise constant DG discretization combined with the forward Euler time stepping. Since our BSGs for this test case have somewhat higher resolution, the used time step is between 4 and 6 seconds compared to the maximum time step of 9 seconds for the original unstructured mesh. The free surface elevation written out at 6 locations (recording stations), Figure 8, is compared to the results produced by the unstructured mesh simulation. For all stations – even at Station 6 in the Sea of Azov and thus the farthest from the open boundary located in the Straits of Gibraltar – the results match well, with the largest difference at Station 3 in the Adriatic Sea. We attribute this difference to a higher
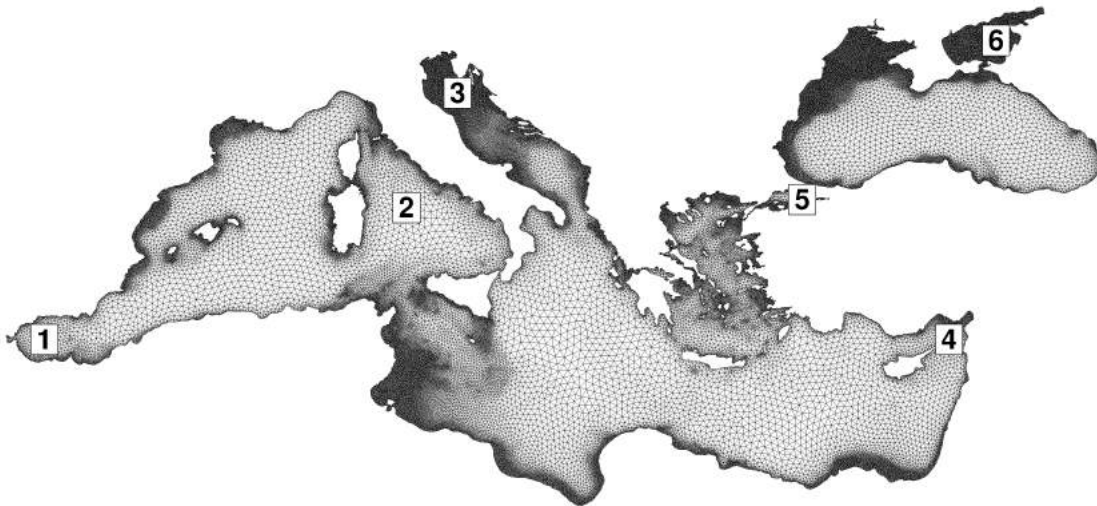
Figure 6: Unstructured mesh for domain Mediterranean with $112\,962$ triangles.

resolution of our BSGs.

The relative number of masked triangles, Equation (9), is low in all presented BSGs and varies between $5\,\%$ and $15\,\%$. A thorough study of the performance of BSGs with masks in simulations, especially in comparison with unstructured meshes, is yet to come.

6. CONCLUSIONS & OUTLOOK

We presented a method for generating block-structured grids which relies on an unstructured triangular mesh as the sole input. The BSGs have a prescribed number of quad blocks refined uniformly into triangular elements. We make use of the Discrete Mesh Optimization (DMO) method for repositioning vertices as we have different quality metrics throughout the method. Masking elements allows representing features much smaller than the block size which is very important when considering complex domain shapes. We evaluate our block-structured grids by comparing them to the unstructured triangular meshes. In future work, we plan to improve the masking process to represent fine details like isthmuses and islands that are even smaller than one element wide. Furthermore, automatic alignment of vertices to interior features will be added.

Acknowledgements

References

[1] Gropp W.D., Kaushik D.K., Keyes D.E., Smith B.F. "Performance Modeling and Tuning of an Unstructured Mesh CFD Application." *SC '00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, pp. 34–34. Nov. 2000

[2] White B.S., McKee S.A., de Supinski B.R., Miller B., Quinlan D., Schulz M. "Improving the computational intensity of unstructured mesh applications." *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, pp. 341–350. Association for Computing Machinery, New York, NY, USA, Jun. 2005

[3] Il'in V. "Integrated Computational Environment for Grid Generation Parallel Technologies." *Parallel Computational Technologies*, Communications in Computer and Information Science, pp. 58–68. Springer International Publishing, Cham, 2020

[4] Bank R.E., Smith R.K. "Mesh Smoothing Using A Posteriori Error Estimates." *SIAM Journal on Numerical Analysis*, vol. 34, no. 3, 979–997, Jun. 1997

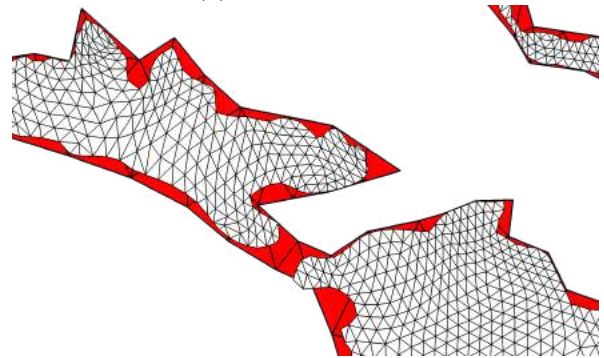[5] Freitag L., Jones M., Plassmann P. "A Parallel Algorithm for Mesh Smoothing." *SIAM Jour-
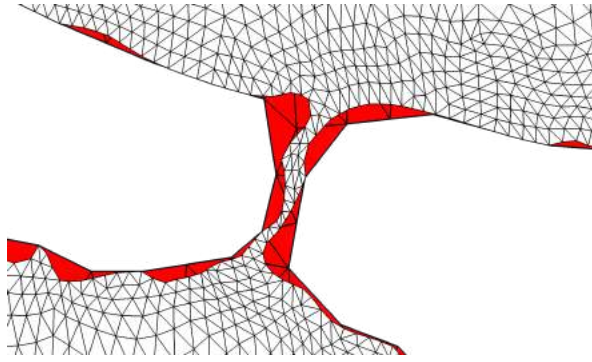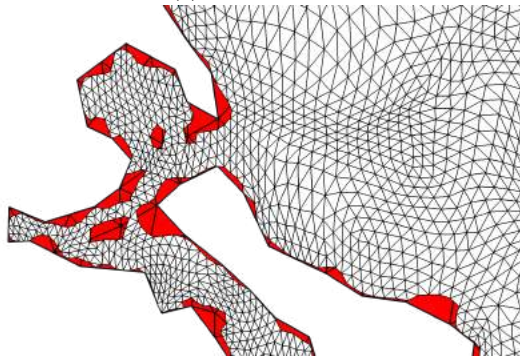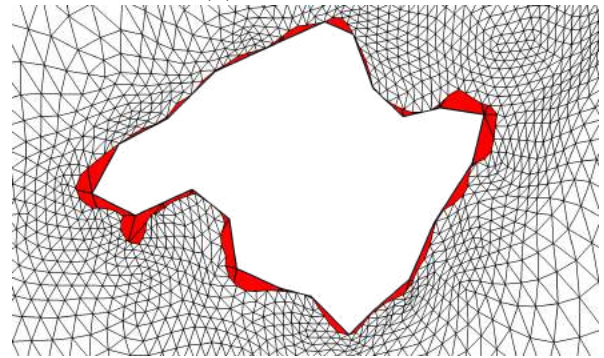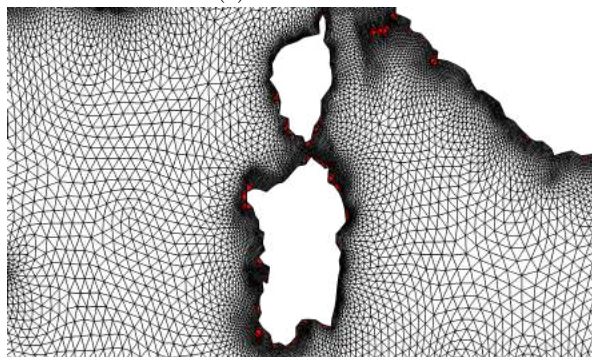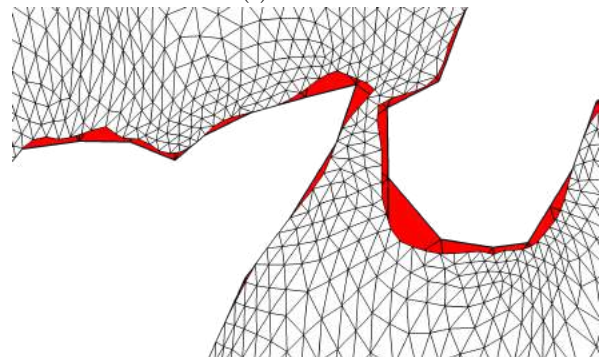
(a) Bosporus
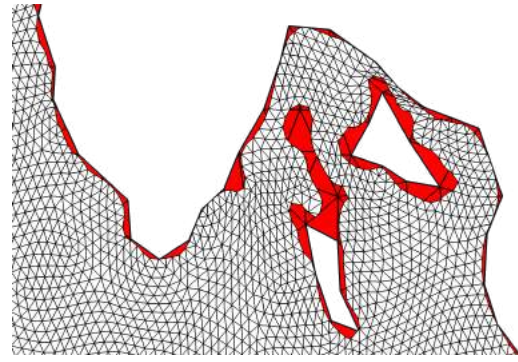
(b) Coast of Croatia
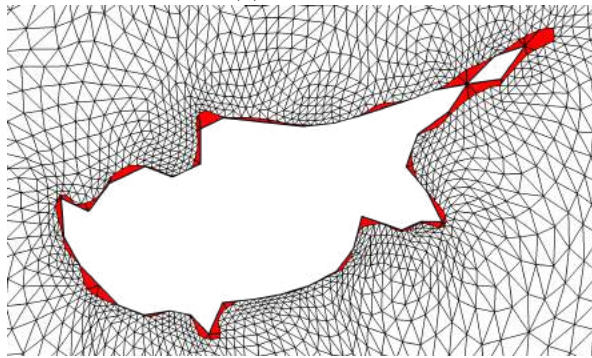
(c) Cyprus

(d) Gulf of Corinth

(e) Malian Gulf

(f) Mallorca

(g) Sardinia

(h) Sicily

Figure 7: Detail views of the Mediterranean domain with $N_B = 8000$. Red regions are covered by the block grid but are not part of the masked BSG.
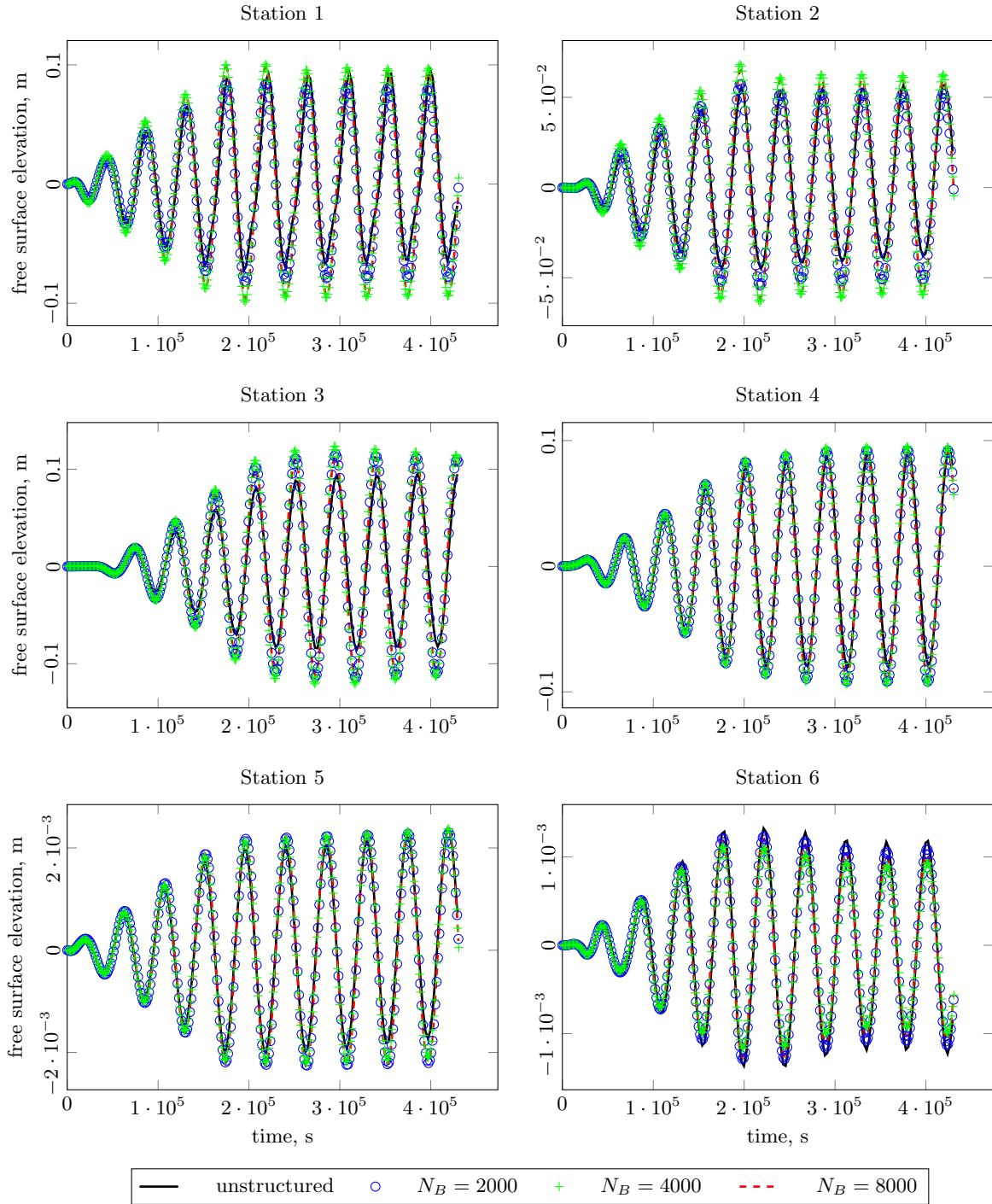
Figure 8: Free surface elevation at recording stations on Mediterranean.

nal on Scientific Computing, vol. 20, no. 6, 2023–2040, Jan. 1999

[6] Bergen B.K., Hülsemann F. "Hierarchical hybrid grids: data structures and core algorithms for

multigrid." *Numerical Linear Algebra with Applications*, vol. 11, no. 2-3, 279–291, 2004

[7] Kohl N., Thönnes D., Drzisga D., Bartuschat D., Rüde U. "The HyTeG finite-element software

framework for scalable multigrid solvers." *International Journal of Parallel, Emergent and Distributed Systems*, vol. 34, no. 5, 477–496, 2019

[8] Turek S., Göddeke D., Becker C., Buijssen S.H., Wobker H. "FEAST—realization of hardware-oriented numerics for HPC simulations with finite elements." *Concurrency and Computation: Practice and Experience*, vol. 22, no. 16, 2247–2265, 2010

[9] Falgout R.D., Jones J.E., Yang U.M. "The design and implementation of hypre, a library of parallel high performance preconditioners." *Numerical solution of partial differential equations on parallel computers*, pp. 267–294. Springer, 2006

[10] Armstrong C.G., Fogg H.J., Tierney C.M., Robinson T.T. "Common themes in multi-block structured quad/hex mesh generation." *Procedia Engineering*, vol. 124, 70–82, 2015

[11] Fogg H.J., Armstrong C., Robinson T.T. "Multi_Block Decomposition Using Cross-Fields." *Proceedings of adaptive modelling and simulation, Lisbon*, pp. 254–267, 2013

[12] Fogg H.J., Armstrong C.G., Robinson T.T. "Automatic generation of multiblock decompositions of surfaces." *International Journal for Numerical Methods in Engineering*, vol. 101, no. 13, 965–991, 2015

[13] Lim C.W., Yin X., Zhang T., Su Y., Goh C.K., Moreno A., Shahpar S. "Automatic Blocking of Shapes Using Evolutionary Algorithm." *International Meshing Roundtable*, pp. 169–188. Springer, 2018

[14] Lim C.W., Yin X., Zhang T., Selvaraj S.K., Su Y., Goh C.K., Moreno A., Shahpar S. "Towards Automatic Blocking of Shapes using Evolutionary Algorithm." *Computer-Aided Design*, vol. 120, 102798, Mar. 2020

[15] Barrera P., Méndez I. "Parametrization of plane irregular regions: A semi-automatic approach I." *Numerical Geometry, Grid Generation and Scientific Computing*, pp. 263–279. Springer International Publishing, Cham, 2021

[16] Xu G., Li M., Mourrain B., Rabczuk T., Xu J., Bordas S.P.A. "Constructing IGA-suitable planar parameterization from complex CAD boundary by domain partition and global/local optimization." *Computer Methods in Applied Mechanics and Engineering*, vol. 328, 175–200, Jan. 2018

[17] Zint D., Grosso R., Aizinger V., Köstler H. "Generation of Block Structured Grids on Complex Domains for High Performance Simulation."

Computational Mathematics and Mathematical Physics, vol. 59, no. 12, 2108–2123, Dec. 2019

[18] Boier-Martin I., Rushmeier H., Jin J. "Parameterization of triangle meshes over quadrilateral domains." *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pp. 193–203. Association for Computing Machinery, New York, NY, USA, Jul. 2004

[19] Carr N.A., Hoberock J., Crane K., Hart J.C. "Rectangular multi-chart geometry images." *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pp. 181–190. Eurographics Association, Goslar, DEU, Jun. 2006

[20] Dong S., Bremer P.T., Garland M., Pascucci V., Hart J.C. "Spectral surface quadrangulation." *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pp. 1057–1066. Association for Computing Machinery, New York, NY, USA, Jul. 2006

[21] Daniels J., Silva C.T., Cohen E. "Semi-regular Quadrilateral-only Remeshing from Simplified Base Domains." *Computer Graphics Forum*, vol. 28, no. 5, 1427–1435, 2009

[22] Campen M. "Partitioning Surfaces Into Quadrilateral Patches: A Survey." *Computer Graphics Forum*, vol. 36, no. 8, 567–588, 2017

[23] Remacle J.F., Lambrechts J., Seny B., Marchandise E., Johnen A., Geuzainet C. "Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm." *International Journal for Numerical Methods in Engineering*, vol. 89, no. 9, 1102–1119, 2012

[24] Tarini M., Pietroni N., Cignoni P., Panozzo D., Puppo E. "Practical quad mesh simplification." *Computer Graphics Forum*, vol. 29, no. 2, 407–418, 2010

[25] Lengauer C., Apel S., Bolten M., Chiba S., Rüde U., Teich J., Größlinger A., Hannig F., Köstler H., Claus L., Grebhahn A., Groth S., Kronawitter S., Kuckuk S., Rittich H., Schmitt C., Schmitt J. "ExaStencils: Advanced Multigrid Solver Generation." *Software for Exascale Computing - SPPEXA 2016-2019*, pp. 405–452. Springer International Publishing, Cham, 2020

[26] Faghih-Naini S., Kuckuk S., Aizinger V., Zint D., Grosso R., Köstler H. "Quadrature-free discontinuous Galerkin method with code generation features for shallow water equations on automatically generated block-structured meshes." *Advances in Water Resources*, p. 103552, 2020

[27] Aizinger V., Dawson C. "A discontinuous Galerkin method for two-dimensional flow and transport in shallow water." *Advances in Water Resources*, vol. 25, no. 1, 67–84, 2002

[28] Garland M., Zhou Y. "Quadric-based simplification in any dimension." *ACM Transactions on Graphics*, vol. 24, no. 2, 209–239, Apr. 2005

[29] Dey T.K., Edelsbrunner H., Guha S., Nekhayev D.V. "Topology Preserving Edge Contraction." *Publ. Inst. Math. (Beograd) (N.S*, vol. 66, 23–45, 1998

[30] Garland M., Heckbert P.S. "Surface simplification using quadric error metrics." *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pp. 209–216. ACM Press/Addison-Wesley Publishing Co., USA, Aug. 1997

[31] Botsch M., Kobbelt L. "A remeshing approach to multiresolution modeling." *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pp. 185–192. Association for Computing Machinery, New York, NY, USA, Jul. 2004

[32] Zint D., Grosso R. "Discrete Mesh Optimization on GPU." *27th International Meshing Roundtable*, Lecture Notes in Computational Science and Engineering, pp. 445–460. Springer International Publishing, Cham, 2019

[33] Bommes D., Lempfer T., Kobbelt L. "Global Structure Optimization of Quadrilateral Meshes." *Computer Graphics Forum*, vol. 30, no. 2, 375–384, 2011

[34] Zint D., Grosso R. "A Hybrid Approach to Fast Indirect Quadrilateral Mesh Generation." *Numerical Geometry, Grid Generation and Scientific Computing*, pp. 281–294. Springer International Publishing, Cham, 2021

[35] Cialone M.A., Militello A., Brown M.E., Kraus N.C. *COUPLING OF WAVE AND CIRCULATION NUMERICAL MODELS AT GRAYS HARBOR ENTRANCE, WASHINGTON, USA*, pp. 1279–1291. World Scientific Publishing Company, 2003