Computing and Visualization in Science (2020) 23:2
https://doi.org/10.1007/s00791-020-00322-5

IMG 2016

Mesh generation for thin layered domains and its application to parallel multigrid simulation of groundwater flow

Sebastian Reiter[1] · Dmitry Logashenko[3] · Andreas Vogel[2] · Gabriel Wittum[1,3]

Received: 31 July 2017 / Accepted: 19 July 2018 / Published online: 20 April 2020

Abstract
The generation of detailed three dimensional meshes for the simulation of groundwater flow in thin layered domains is crucial to capture important properties of the underlying domains and to reach a satisfying accuracy. At the same time, this level of detail poses high demands both on suitable hardware and numerical solver efficiency. Parallel multigrid methods have been shown to exhibit near optimal weak scalability for massively parallel computations of density driven flow. A fully automated parameterized algorithm for prism based meshing of coarse grids from height data of individual layers is presented. Special structures like pinch outs of individual layers are preserved. The resulting grid is used as a starting point for parallel mesh and hierarchy creation through interweaved projected refinement and redistribution. Efficiency and applicability of the proposed approach are demonstrated for a parallel multigrid based simulation of a realistic sample problem.

Keywords Grid generation · Layered domain · Groundwater flow · Geometric multigrid method · Parallelization

1 Introduction

While highly efficient approaches to parallel multigrid methods for large parallel computers have been developed in recent years (cf. [1,2,4,11,17,18]), the efficient application of those approaches to realistic problem settings can still be challenging. Considering the large time frames in which effects of ground water flow may influence, e.g., the transport of nuclear waste, an efficient realization of each time step is of crucial importance.

To fully resolve the effects of groundwater flow on realistic three dimensional layered domains with large horizontal extensions, meshes with huge element numbers are required. However, due to the specific geometry of such domains, which often feature thin layers of different materials and jumping coefficients stacked above each other, meshing with good element qualities is a major concern for numerical simulations.

In [17] we showed that our approach to massively parallel geometric multigrid in our simulation software *UG4* [19] is highly scalable for up to hundred thousands of processes. Its applicability to density driven flow equations was shown in [12]. Again, we observed very good scalability for all tested process numbers.

In this article, we describe a meshing and hierarchy creation technique for distributed three dimensional layered domains with special focus on its eligibility for parallel geometric multigrid methods. Work in this area regarding meshing and multigrid simulation of density driven flow in layered domains has been done in previous works (cf. [8,12,13]). However, the generation of adaptive surface nets and the quality of the geometric approximation in higher grid

Communicated by Babett Lemke.

✉ Sebastian Reiter
sreiter@gcsc.uni-frankfurt.de

Dmitry Logashenko
dmitry.logashenko@kaust.edu.sa

Andreas Vogel
a.vogel@rub.de

Gabriel Wittum
gabriel.wittum@kaust.edu.sa

1 Goethe Center for Scientific Computing (G-CSC), Goethe University Frankfurt am Main, Kettenhofweg 139, 60325 Frankfurt, Germany

2 High Performance Computing in the Engineering Sciences, Ruhr University Bochum, Universitätsstraße 150, 44801 Bochum, Germany

3 Computer, Electrical and Mathematical Sciences and Engineering Division, King Abdullah University of Science and Technology, 4700 KAUST, Al-Khawarizmi (Bldg. 1, West), Thuwal, Jeddah 23955-6900, Kingdom of Saudi Arabia

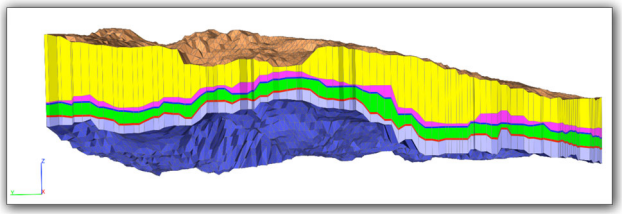

Fig. 1 Cut through a layered domain with 6 layers, illustrating the variance in height of individual and neighbored layers (the plot is scaled by a factor 50 in the vertical direction)

levels have not been considered in those works and we thus focus on these aspects in the present paper.

To this end, we briefly describe the underlying mathematical model and numerical methods used in Sect. 2. The construction of a coarse grid from height values of individual layers is described in Sect. 3. The resulting grid resolves all important characteristics of the considered domain. Starting from this coarse grid, an interweaved anisotropic refinement and redistribution scheme is given in Sect. 4, through which successively more and more elements are created and more and more processes are used. During refinement, we thereby position new vertices using a special projector, which is constructed from the original height data of the individual layers. The resulting hierarchy thus approximates the original domain better with each additional refinement step. Finally, we apply this algorithm to a test problem on a realistic domain in Sect. 5.

2 Underlying model and numerics

We consider density driven flow in domains with large horizontal and small vertical extensions. While layers may stretch across 10,000 km^2, they are as thin as 1 m or may even pinch out and vanish completely locally. Several such layers with vastly differing permeabilities are stacked upon each other and form a larger domain with a total thickness of 10–100 m or more. Figure 1 shows a cut through a reconstructed domain, illustrating how the height of individual layers changes and how the height between neighbored layers varies. The geometry in that image is scaled along the z-axis by a factor of 50, so that all layers are clearly visible.

2.1 Model

The underlying mathematical model for density driven flow is comprised of two nonlinear, coupled, and time dependent differential equations:

$$\partial_t(\phi\varrho) + \nabla\cdot(\varrho q) = 0$$
$$\partial_t(\phi\varrho\omega) + \nabla\cdot(\varrho\omega q - \varrho D\nabla\omega) = 0$$

with

$$q = -\frac{K}{\mu}(\nabla p - \varrho g)$$

where the unknowns ω and p are the mass fraction of salt and the hydrodynamic pressure respectively, ϕ is the porosity, ϱ the density of the fluid phase, K the permeability, μ the viscosity, g the gravity field, and D the diffusion–dispersion tensor (cf. [14]). The equations describe the balance of the fluid phase as a whole and the balance of the mass of the brine.

2.2 Discretization

Discretization of this model is performed using a mass preserving, node-centered finite volume method (cf. [10]). Regarding the discretization of the underlying domain, special care has to be taken for mesh generation due to the presence of very thin horizontal layers.

Mesh generation with tetrahedral elements would either lead to a vast amount of elements or to very small or very large inner angles in elements in thin layers. Large element numbers have a negative impact on solver efficiency and do not play well with a geometric multigrid approach, where the generated grid is used as the coarse grid of the hierarchy. On the other hand, fewer elements with very high or very low inner angels deteriorate the condition of the stiffness matrix and may lead to convergence issues in the solver (cf. [9]).

Prism and hexahedral meshes have the benefit of allowing for the meshing of thin layers with a comparatively small number of elements, each with good inner angles ($0° \ll \alpha \ll 180°$). Furthermore, they have the advantage that special refinement techniques can be used to improve element aspect ratios in higher levels of the mesh hierarchy (cf. [3,8]). Those properties align nicely with the requirements of a geometric multigrid solver. For the presented mesh and hierarchy creation approach, prisms provide both the necessary flexibility and structuredness to build an efficient simulation setup.

2.3 Solver

In [17] we showed that the geometric multigrid method and its implementation in *UG4* scales perfectly fine to hundred thousands of processes if certain measures regarding the distribution of lower levels are taken. The basic approach is to start from a coarse grid and successively generate finer and finer levels through repeated refinement. In each refinement step, we also compute an optimal distribution of the hierarchy and, if necessary, redistribute the elements among all processes. Using a so called *process hierarchy*, we make

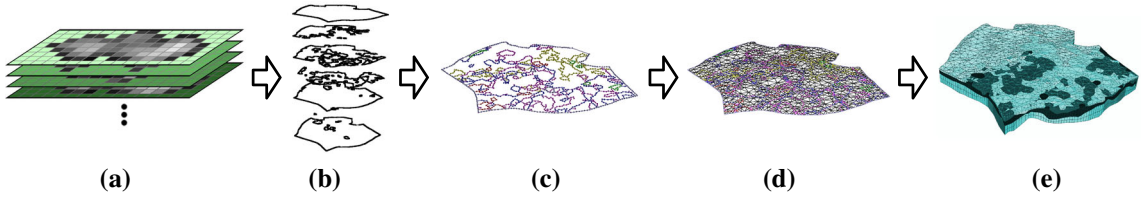

Fig. 2 Meshing sequence from fine raster data to coarse prism mesh. Green cells in the raster data represent *no-data* values. Depicted are, from left to right: (a) Input raster stack → (b) boundary extraction → (c) simplified boundary mesh → (d) triangulated surface mesh → (e) prism mesh

sure that the computation to communication ratio is fine on all levels [16].

The solution is finally computed on the finest level of the generated hierarchy, using a BiCGSTAB method preconditioned by a geometric multigrid method. The latter uses coarser levels to speed up computations by removing lower error frequencies through a coarse grid correction.

We successively applied this method to parallel benchmark problems of density driven flow, too (cf. [12]). However, the grid complexity in applications like the one presented here requires several specific adjustments in the meshing and redistribution phases.

3 Coarse grid generation

The role of the coarse grid is fundamental in a geometric multigrid method. It should contain as few elements as possible while still providing the topology, i.e. connectivity of subdomains, of the given domain and a reasonable geometric approximation to all subdomains.

In the following, we describe our coarse grid meshing approach to a layered domain which is specified as a set of layer interfaces. See Fig. 2 for a graphical overview of the meshing sequence.

The whole meshing algorithm has been realized in the software ProMesh (cf. [15]).

3.1 Input data

For a domain consisting of L soil layers S^l, $0 \leq l < L$, the input data to our coarse grid meshing algorithm is a set of $L+1$ equidistant, regular 2d grids. Without loss of generality we assume that all grids have the same number of rows $n_r \in \mathbb{N}$ and columns $n_c \in \mathbb{N}$ and that each cell has the width and height of 1. With each cell, we associate a value $h^l_{i,j} \in \mathbb{R}$, $0 \leq i < n_r$, $0 \leq j < n_c$. The values $h^l_{i,j}$ of layer l, $0 \leq l < L+1$ are interpreted as absolute height values describing the lower boundary of the soil layer S^l. The upmost field h^0 is interpreted as the surface elevation. We furthermore define an invalid value $\bar{h}^l$ which marks an entry as a *no-data* entry:

$$h^l_{i,j} = \bar{h}^l \Rightarrow \text{layer } S^l \text{ is non-existent at } (i, j).$$

This value is set to a height which is far above or below the modeled region.

Without loss of generality we assume that the south west corner of the modeled region is located in the origin $(0, 0)$. Only a simple coordinate transformation is required to extend this setting to the general case and to associate meaningful physical units.

Figure 2a depicts a stack of such regular 2d grids, where gray values represent the value $h^l_{i,j}$ (height) of each cell and green represents no-data entries.

3.2 Preprocessing

We allow for the specification of a *minimal height* parameter $h^l_{min} \in \mathbb{R}$, $h^l_{min} > 0$ for each layer. The height $H^l_{i,j} \in \mathbb{R}$ of layer $S^l, 0 \leq l < L$ at a the index pair i, j shall be determined by

$$H^l_{i,j} := \min \left\{ h^l_{i,j} - h^k_{i,j} \mid l < k \leq L,\ h^k_{i,j} \neq \bar{h} \right\} \cup \{0\}.$$

Iterating from $l = L-1, L-2, \ldots, 0$, we then check whether $H^l_{i,j} < h^l_{min}$. If this is the case, we consider the associated layer S_l to be non-existent at that position and we set the corresponding entry $h^l_{i,j}$ to $\bar{h}$.

Please note that the presence of *no-data* values means that individual soil layers are not necessarily connected.

3.3 The grid data structure

In the following, we describe how we generate a *finite element mesh* from the input data. To this end, we use the following notion. A mesh or *grid* G consists of a set of *vertices* V_G, a set of edges E_G, a set of *faces* F_G (e.g. triangles, quadrilaterals), and a set of *volumes* O_G (e.g. tetrahedra, hexahedra, prisms, pyramids): $G := V_G \cup E_G \cup F_G \cup O_G$.

Each element $e \in E_G \cup F_G \cup O_G$ shall thereby be uniquely identified by its set of corner vertices $v_1, \ldots, v_n \in V_G$. If $v \in V_G$ is a corner of an element $e \in G$, we write $v \in e$. Similarly, if $e_1 \in G$ is a side or a side of a side of $e_2 \in G$, we write $e_1 \in e_2$.

For the grid G we introduce mappings $p_x, p_y, p_z, p_w \to \mathbb{R}$, and $p : V_G \to \mathbb{R}^4$ such that for every $v \in V_G$, $p(v) := (p_x(v),\ p_y(v),\ p_z(v),\ p_w(v))$, where $(p_x(v), p_y(v), p_z(v))$

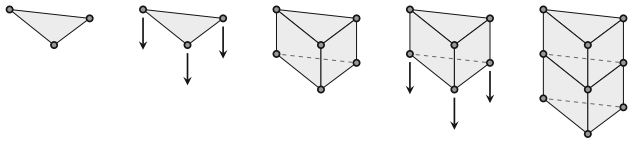

Fig. 3 Repeated extrusion of a triangle into a stack of prisms

is the geometric position of v and $p_w(v)$ is an auxiliary value used during meshing and for projected refinement.

3.4 Boundary meshing

In the first meshing step we extract a set of edge paths, representing the boundaries of each layer, from the raster data. To this end, for a soil layer S^l, we consider cells of neighboring height values $c_{i,j} := (h_{i,j}, h_{i+1,j}, h_{i,j+1}, h_{i+1,j+1})$ for $0 \leq i < n_r - 1$, $0 \leq j < n_c - 1$. For each cell $c_{i,j}$, $0 \leq i < n_r - 1$, $0 \leq j < n_c - 1$, we perform one of the following operations, depending on the number of *no-data* entries in $c_{i,j}$:

- **0 *no-data* entries**: Check neighboring cells $c_{i\pm1,j}$ and $c_{i,j\pm1}$. If one of them contains 2 or more *no-data* entries then add the corresponding separating edge to the grid G. We thereby consider a cell $c_{k,l}$ with $k < 0$, $k >= n_r - 2$, $l < 0$, or $l > n_c - 2$ as a cell with 4 *no-data* entries,
- **1 *no-data* entry**: find the diagonal which does not contain the *no-data* entry and add it to the grid G,
- **2, 3, or 4 *no-data* entries**: ignore the cell.

When adding an edge to a grid G we first add two vertices to the mesh, one for each corner point, and then add an edge which connects those two vertices. We thereby use the index-pairs of the corner points as $2d$ coordinates of the respective vertices.

Once all cells of each layer have been processed, we resolve self intersections of the resulting grid. To this end, we merge different vertices with identical positions into one vertex. Furthermore, if for two edges $e_1, e_2 \in E_G$ the intersection $e_1 \cap e_2$ is not empty, we introduce a new vertex at their intersection and split both e_1 and e_2 according to the new vertex.

3.5 Boundary simplification

Depending on the resolution of the provided raster data, the extracted boundary mesh can be very fine. To avoid unnecessary detail during coarse grid construction, we replace two neighbored edges $e_1, e_2 \in E_G$ which share a common vertex $v \in V_G$, if the following conditions are met:

- e_1 and e_2 are the only edges which are connected to v.
- $\angle_{s,i}(e_1, e_2) < \alpha$, where $0 < \alpha < \pi$ is a user supplied threshold angle.
- $|e_1| < \frac{maxLen}{2}$, $|e_2| < \frac{maxLen}{2}$.

Here, $\angle_{s,i} : E_G \times E_G \to \mathbb{R}$ with $s \in (0, 1)$, $i \in \mathbb{N}^+$, denotes the angle between two edges on an i-times smoothed copy of the underlying grid G. For $v \in V_G$ let $[v, v_1], \ldots, [v, v_n]$ with $v_1, \ldots, v_n \in V_G$ be the set of edges which contain v. One smoothing step for v is then performed by setting

$$p(v) = (1 - s) \cdot p(v) + \frac{s}{n} \sum p(v_n).$$

One smoothing step for the whole grid G is performed by applying one smoothing step to all its vertices.

Note, that the smoothing does not affect the positions of the simplified boundary representation. It is only considered to evaluate whether nodes should be removed during simplification.

3.6 Surface triangulation

Based on the resulting edge mesh we first perform a *sweep-line* triangulation (cf. [7]) followed by *constrained delaunay retriangulation* on G (cf. [5]). This yields a coarse triangle mesh with good triangle qualities. Furthermore, all edges of the boundary representation are contained in the resulting mesh as sides of triangles.

3.7 Extrusion

Starting from a flat triangulation we generate a prism grid by repeatedly extruding the bottom triangles of the current grid, by calling $Extrude\,(G,\ F_G,\ V_G,\ 1,\ L)$ from Algorithm 1, cf. Fig. 3. Prior to calling $Extrude$, we set the w-coordinate of all vertices $v \in V_G$ to 0.

```
Extrude (G, T, V, i, imax) {
 Copy vertices V -> V';
 Set i as w-coordinate to all vertices in V';
 Copy T -> T', using V' as corner-vertices;
 Connect corresponding triangles in T and T' by prisms;
 Call Extrude (G, T', V', i+1, imax);
}
```

Algorithm 1: Extrusion algorithm for a grid G, a set of triangles T, the set of corner vertices V of T, a counter i, and the number of consecutive extrusions *imax*. All new elements are created in the specified grid G.

Note that the x- and y-coordinate of each vertex are determined by the surface triangulation. During extrusion we

furthermore assign a w-coordinate. The z-coordinate has not yet been assigned. The w-coordinate has no physical meaning. It is an auxiliary value which will be used to get the corresponding physical z-coordinate using a special mapping as described below. The auxiliary value is furthermore used during parallel projected refinement as detailed in Sect. 4.

3.8 Adjustment of height values

Each layer of triangles in the extruded geometry corresponds to one of the heightfields $h^l, 0 \leq l < L$. During *extrusion*, we associated a w-coordinate with each vertex, corresponding to the index of the corresponding heightfield h^l. A straight forward idea to compute the corresponding z-coordinate of a vertex $v \in V_G$ would be to use a lookup in the heightfield h^l, where $l = p_w(v)$. However, as described in the Sect. *Preprocessing*, the heightfields h^l may contain *no-data* entries.

To still allow for a simple lookup, we thus create a second set of heightfields $\tilde{h}^l$, $0 \leq l < L+1$, in which we fill the *no-data* holes with meaningful intermediate height values. We start by defining for each layer the absolute height of its upper and lower interface: For $0 < l \leq L$, let

$$\hat{l}^l_{i,j} := max\left\{k \in \mathbb{N} \mid k < l,\ h^k_{i,j} \neq \bar{h}\right\}$$

be the index of the next heightfield above layer l, where a valid value is defined at i, j. Similarly, for $0 \leq l \leq L$ let

$$\check{l}^l_{i,j} := min\left\{m \in \mathbb{N} \mid m \geq l,\ h^m_{i,j} \neq \bar{h}\right\}$$

be the index of the next heightfield at or below layer l, where a valid value is defined at i, j.

We initialize $\tilde{h}$ by simply interpolating values from upper and lower valid values:

$$\tilde{h}^l_{i,j} := \begin{cases} h^l_{i,j} & \text{if } h^l_{i,j} \neq \bar{h}, \\ \frac{m-l}{m-k} h^k_{i,j} + \frac{l-k}{m-k} h^m_{i,j} & \text{else,} \end{cases}$$

where $k = \hat{l}^l_{i,j}$ and $m = \check{l}^l_{i,j}$, depending on i, j. By construction it holds: $\tilde{h}^l_{i,j} > \tilde{h}^{l+1}_{i,j}$.

Since simple interpolation may introduce unwanted steep gradients and bad angles, we smoothen newly introduced height values. Here it is crucial to avoid unnaturally thin or even flipped elements. To this end we must keep the property $\tilde{h}^l_{i,j} > \tilde{h}^{l+1}_{i,j}$. Instead of smoothing height values directly, we thus prefer to smoothen distance ratios between a value and its upper/lower layer. Let l, i, j be indices such that $h^l_{i,j} = \bar{h}$, $0 < l < L$, and $\tilde{h}^{l-1} \neq \tilde{h}^{l+1}$. First we define the distance ratio

$$r^l_{i,j} := \frac{\tilde{h}^{l-1}_{i,j} - \tilde{h}^l_{i,j}}{\tilde{h}^{l-1}_{i,j} - \tilde{h}^{l+1}_{i,j}}.$$

Fig. 4 Creation of smoothed height lookup tables $\tilde{h}^l$ (2d slice). Left: initial height values. Edges to and between *no-data* values are not shown. Middle: *no-data* values are replaced through simple interpolation between upper/lower layer. Right: smoothing applied to new height values

In one smoothing iteration we then perform for each such triple l, i, j the following smoothing step:

$$\tilde{h}^l_{i,j} := \tilde{h}^{l-1}_{i,j} + \left(\tilde{h}^{l-1}_{i,j} - \tilde{h}^{l+1}_{i,j}\right) \cdot \Big[(1-\alpha) \cdot r^l_{i,j} + \alpha \cdot \left(r^l_{i-1,j} + r^l_{i+1,j} + r^l_{i,j-1} + r^l_{i,j+1}\right)\Big],$$

where $\alpha \in (0,1)$ is a user defined smoothing constant. Figure 4 shows an illustration of the construction of $\tilde{h}$.

Once the smoothed field is created, we introduce bilinear interpolation ϕ^l on the raster values for each smoothed layer $\tilde{h}^l$, $0 \leq l \leq L$:

$$\begin{aligned} &\phi^l : [0, n_c] \times [0, n_r] \to \mathbb{R} \\ &\phi^l(x, y) := (\lceil x \rceil - x) \cdot (\lceil y \rceil - y) \cdot \tilde{h}^l_{\lfloor x \rfloor, \lfloor y \rfloor} \\ &\quad + (x - \lfloor x \rfloor) \cdot (\lceil y \rceil - y) \cdot \tilde{h}^l_{\lceil x \rceil, \lfloor y \rfloor} \\ &\quad + (\lceil x \rceil - x) \cdot (y - \lfloor y \rfloor) \cdot \tilde{h}^l_{\lfloor x \rfloor, \lceil y \rceil} \\ &\quad + (x - \lfloor x \rfloor) \cdot (y - \lfloor y \rfloor) \cdot \tilde{h}^l_{\lceil x \rceil, \lceil y \rceil}. \end{aligned}$$

Using this mapping we then define a $3d$ mapping Φ for the whole domain by interpolating between neighboring layers:

$$\begin{aligned} &\Phi : [0, n_c] \times [0, n_r] \times [0, L] \to \mathbb{R} \\ &\Phi(x, y, w) := \beta \cdot \phi^{\lfloor w \rfloor + 1}(x, y) + (1 - \beta) \cdot \phi^{\lfloor w \rfloor}(x, y) \end{aligned}$$

with $\beta := w - \lfloor w \rfloor$.

For each vertex $v \in V_G$ we then assign the z-coordinate (the absolute height value)

$$p_z(v) := \Phi(p_x(v),\ p_y(v),\ p_w(v)).$$

4 Parallel projected refinement

To create a distributed grid hierarchy suitable for efficient geometric multigrid methods, we perform repeated distribution and refinement of the hierarchy similar to [16]. The main steps will be outlined below.

4.1 Refinement

Refinement of an element $e \in G$ denotes the subdivision of e into new elements $e_1, \ldots, e_n$, as depicted in Fig. 5 for

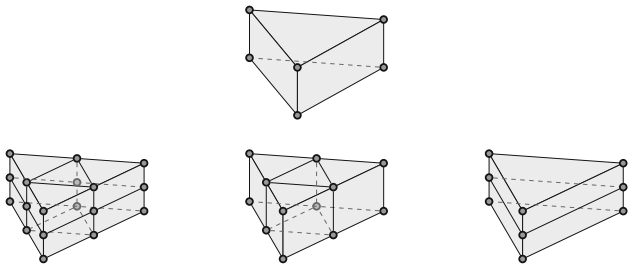

Fig. 5 Refinement of a prism. **Top:** initial prism. **Left:** *isotropic* refinement. **Middle:** *anisotropic* refinement. **Right:** *vertical* refinement

prism elements. We call e the parent element of the elements $e_1, \ldots, e_n$. In the context of geometric multigrid methods, refinement is crucial to construct a sufficiently fine grid from the initial coarse grid. Considering hierarchy creation, refinement of a grid G_k yields a finer grid G_{k+1}, which is used as the next level in a multigrid hierarchy. Starting with the coarse grid G_0, repeated refinement thus results in a grid hierarchy $G_0, G_1, \ldots, G_K$, $K \in \mathbb{N}$.

To improve the bad element aspect ratios present due to the high anisotropies of the different layers, we perform a mixture of anisotropic and regular prism refinement (cf. Fig. 5). The most important refinement types are

- ***Isotropic refinement*** (*'i'*): A prism is subdivided into 8 smaller prisms, each similar to the original (cf. Fig. 5 left),
- ***Anisotropic refinement*** (*'a'*): A prism is subdivided into 4 smaller prisms, improving aspect ratios of flat prisms by a factor of 2 (cf. Fig. 5 middle),
- ***Vertical refinement*** ('v'): Only vertical edges of a prism are being refined. This cuts a prism into 2 smaller prisms, deteriorating aspect ratios of flat prisms (cf. Fig. 5 right).

A refinement sequence can then be specified, e.g. like this: *'-refSeq i2a2i'*, which means that first two isotropic refinements shall be performed, followed by 2 anisotropic refinements, followed by an arbitrary amount of isotropic refinements.

In this article we only consider global refinement, i.e. all elements of the upmost layer are refined at the same time.

4.2 Projection

At each refinement step new vertices are introduced. For each new vertex we have to assign a position. We do this by first placing each new vertex v^* at the arithmetic mean of the corners of its parent element (regarding x-, y-, and w-coordinate) and then interpolate the z-coordinate using:

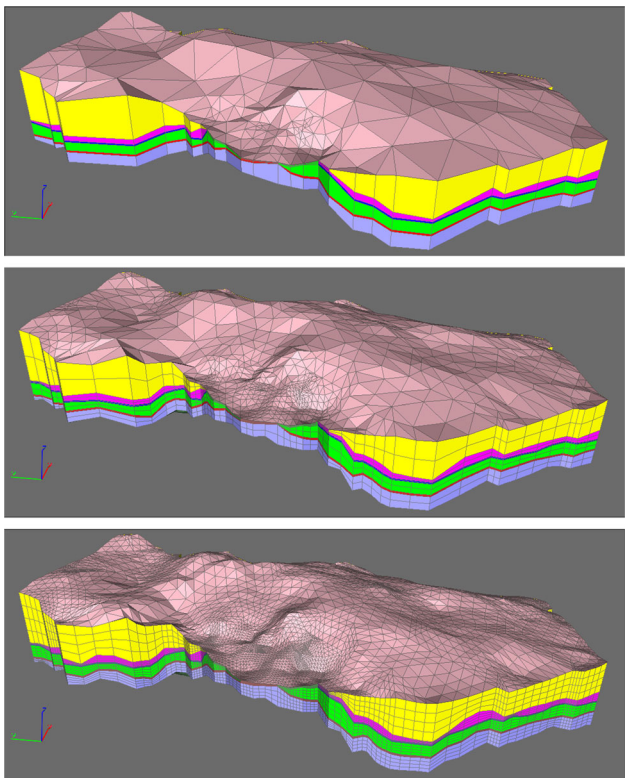

Fig. 6 Three levels resulting from projected refinement. The geometric approximation of individual layers improves as the grid is refined (plots are scaled by a factor 50 in the vertical direction)

$$p_z(v^*) := \Phi\left(p_x(v^*),\ p_y(v^*),\ p_w(v^*)\right).$$

New vertices appearing at layer interfaces are thus placed exactly in the corresponding interface again. New vertices appearing inside a layer are interpolated according to their relative distance to the next upper and the next lower interface as defined by the mapping Φ. By construction of Φ, the interpolation does not lead to flipped or degenerated elements.

Three levels of a resulting hierarchy are shown in Fig. 6. While layers in the coarsest level are not very detailed, finer levels clearly provide more geometrical detail thanks to the described projection approach.

The generated hierarchy is no longer nested. If a vertex has a parent-edge, -face, or -volume, it is not necessarily geometrically contained in the parent element. In our experiments, the non-nestedness did not have a negative effect on the convergence of the multigrid method, even with standard algebraic prolongation and restriction operations. Since the main focus of this paper is on parallel grid generation, a detailed mathematical analysis of this empirical behavior is left to future works.

4.3 Parallelization and hierarchy creation

During hierarchy creation in a simulation, the number of volume elements grows by a certain factor with each refinement step (8 for isotropic refinement, 4 for anisotropic refinement). The number of elements quickly becomes too big to be handled by a single processor. Parallelization is thus a necessity to handle grid hierarchies of such complexity in 3d. In [12,17] we outlined our approach to massively parallel geometric multigrid on hierarchically distributed domains and its application to density driven flow. We use a similar setup for the computations performed on the presented layered domains.

Typically, we start by distributing the base grid to a certain number of processes (e.g. 16) to allow for the application of a parallel base solver. We then perform a certain number of projected refinements in parallel and redistribute again, this time on a larger number of processes. To maintain a good computation to communication ratio, we do not distribute the lower levels to new processes. Instead, only the top-level is redistributed. From here we perform further parallel projected refinement and eventually redistribute further.

To be able to efficiently perform parallel projected refinement, we store the mapping Φ on all involved processes. Unless this mapping is too detailed, this is a sufficiently efficient approach. Several optimizations are possible, if this step would take too much time. One could, e.g., slice the mapping Φ in a similar way in which the grid is partitioned, and only send relevant parts of Φ along with grid partitions.

As noted in [13], solver convergence may require that vertical stacks of prisms are completely contained on one process. This has to be considered as a constraint during grid partitioning.

5 Application

As a benchmark problem we considered a large scale simulation of density driven flow based on the *WIPP* problem described in [6]. We used a simplified model consisting of 6 layers with constant permeability coefficients per layer ranging from 10^{-17} to 10^{-12}.

Aiming for a robust core simulation setup, we focused on parallel meshing, load balancing, and efficient solver setup. For our benchmark simulation we chose simplified Dirichlet boundary conditions $c = 0$, and $p = 0$ on the upper surface of the geometry, and $c = 1$ on the bottom of the geometry. We performed a simulation over a time-frame of 20,000 years, using an *Implicit Euler* method. The arising non-linear equations were solved with the *Newton* method. Inversion of the linear operator was performed using a *BiCGSTAB* solver preconditioned by a parallel *V-Cycle Geometric Multigrid Method* with 5 *ILU pre-* and *post-smoothing* steps and a parallel *ILU* preconditioned *BiCGSTAB* base solver. The simulation was performed using the *UG4* simulation framework (cf. [19]).

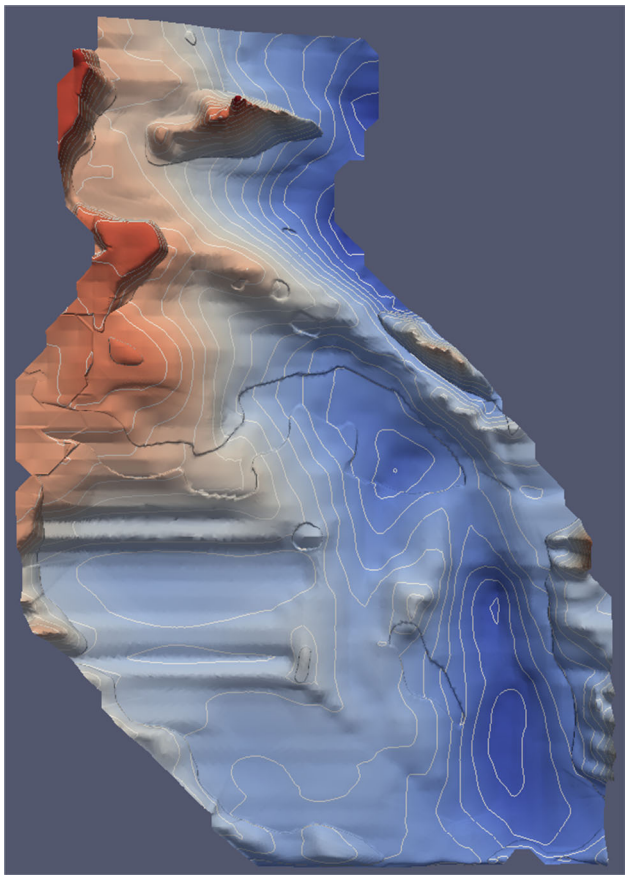

Fig. 7 Isosurface of the concentration of the benchmark problem for $c = 0.05$ after 20,000 years. Contour lines indicate the elevation of the isosurface

The coarse mesh for our simulation is depicted at the top of Fig. 6. Since this mesh does not contain any inner degrees of freedom inside the individual layers, we first performed 3 vertical refinements ('v3'). While increasing the anisotropy of the underlying grid, this improves the approximation of the solution in the individual layers. The resulting top level (level 4) was then used as the base level for the geometric multigrid solver. Starting from this base level we then performed four isotropic and one anisotropic refinements ('i4a1'), yielding a total of 3×10^8 unknowns in the final top level.

The base level (level 4) was distributed to 16 processes, level 6 to 512 processes, and level 8 to 4096 processes. We performed the simulation with time step sizes of 50 years. Fig. 7 shows a concentration profile of the computed distribution after 20,000 years.

6 Conclusions

We presented an algorithm for the construction of coarse grids which are suited for the creation of distributed multigrid hierarchies with a good geometric approximation to the underlying domain. The applicability of our algorithm was demonstrated in a parallel simulation of density driven flow in a complex layered domain over a time frame of 20,000 years.

As noted, other authors proposed extrusion based meshing techniques for the creation of layered domains, too (cf. [8]). New in our approach is the automatic creation of an unstructured triangular net which adheres to pinch-outs of individual layers and which is used as a base for an extrusion process which results in a semi structured prism mesh suitable for anisotropic refinement. During the meshing process we also construct a map which is used to place new vertices during parallel multigrid hierarchy creation, resulting in a hierarchy which resembles the original domain better with each refinement step. This allowed for the construction of a coarse grid with comparatively few elements while still tightly resembling the underlying domain in higher levels.

Acknowledgements This work has been supported by the *German Ministry of Economics and Technology (BMWi,* 02E11476B) and by the DFG Priority Program 1648 *Software for Exascale Computing* (SPPEXA) in the project *Exasolvers* (WI 1037/24-2). We thank the HLRS for the opportunity to use *Hazel Hen* and their kind support. The authors also gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUQUEEN at Jülich Supercomputing Centre (JSC).

References

1. Baker, A., Falgout, R., Kolev, T., Yang, U.: Multigrid smoothers for ultra-parallel computing. SIAM J. Sci. Comput. **33**, 2864–2887 (2011)
2. Bastian, P., Blatt, M., Scheichl, R.: Algebraic multigrid for discontinuous galerkin discretizations of heterogeneous elliptic problems. Numer. Linear Algebra Appl. **19**(2), 367–388 (2012)
3. Bastian, P., Wittum, G.: Adaptive multigrid methods: the UG concept. In: Notes on Numerical Fluid Mechanics, vol. 46, pp. 17–17 (1994)
4. Bergen, B., Gradl, T., Rude, U., Hulsemann, F.: A massively parallel multigrid method for finite elements. Comput. Sci. Eng. **8**(6), 56–62 (2006)
5. Chew, L.P.: Constrained delaunay triangulations. Algorithmica **4**(1), 97–108 (1989)
6. Corbet, T., Knupp, P.: The role of regional groundwater flow in the hydrogeology of the Culebra Member of the Rustler Formation at the Waste Isolation Pilot Plant (Wipp), Southeastern New Mexico. University of North Texas Libraries, Tech. rep. (1996)
7. de Berg, M., Cheong, O., Van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Santa Clara (2008)
8. Feuchter, D.: Geometrie- und gittererzeugung für anisotrope schichtengebiete. Ph.D. thesis, Universität Heidelberg (2008)
9. Field, D.A.: Qualitative measures for initial meshes. Int. J. Numer. Methods Eng. **47**(4), 887–906 (2000)
10. Frolkovič, P., De Schepper, H.: Numerical modelling of convection dominated transport coupled with density driven flow in porous media. Adv Water Resour. **24**(1), 63–72 (2000)
11. Gmeiner, B., Köstler, H., Stürmer, M., Rüde, U.: Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. Concurr. Comput. Pract. Exp. **26**(1), 217–240 (2014)
12. Heppner, I., Lampe, M., Nägel, A., Reiter, S., Rupp, M., Vogel, A., Wittum, G.: Software framework ug4: parallel multigrid on the hermit supercomputer. In W. E. Nagel, D. H. Kröner, & M. M. Resch (Eds.), High performance computing in science and engineering '12: transactions of the High Performance Computing Center, Stuttgart (HLRS), 435–449 (2013)
13. Johannsen, K.: Numerische aspekte dichtegetriebener strömung in porösen medien. Habilitation (2004)
14. Hassanizadeh, S.M., Leijnse, T.: On the modeling of brine transport in porous media. Water Resour. Res. **24**(3), 321–330 (1988)
15. Promesh: http://www.promesh3d.com. Accessed June 2017
16. Reiter, S.: Effiziente algorithmen und datenstrukturen für die realisierung von adaptiven, hierarchischen gittern auf massiv parallelen systemen. Ph.D. thesis, Universität Frankfurt am Main (2014)
17. Reiter, S., Vogel, A., Heppner, I., Rupp, M., Wittum, G.: A massively parallel geometric multigrid solver on hierarchically distributed grids. Comp. Vis. Sci. **16**(4), 151–164 (2013)
18. Sundar, H., Biros, G., Burstedde, C., Rudi, J., Ghattas, O., Stadler, G.: Parallel geometric–algebraic multigrid on unstructured forests of octrees. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp. 43:1–43:11. IEEE Computer Society Press, Los Alamitos, CA (2012)
19. Vogel, A., Reiter, S., Rupp, M., Nägel, A., Wittum, G.: UG 4: a novel flexible software system for simulating PDE based models on high performance computers. Comp. Vis. Sci. **16**(4), 165–179 (2013)