Computers & Geosciences 86 (2016) 83–91

Contents lists available at ScienceDirect

Computers & Geosciences

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

Case study

Rasterizing geological models for parallel finite difference simulation using seismic simulation as an example

Björn Zehner [a,*], Olaf Hellwig [b], Maik Linke [b], Ines Görz [b], Stefan Buske [b]

[a] Federal Institute for Geosciences and Natural Resources (BGR), Wilhelmstraße 25-30, 13593 Berlin, Germany
[b] Institute of Geophysics and Geoscience Informatics, TU Bergakademie Freiberg, Gustav-Zeuner-Str. 12, 09599 Freiberg, Germany

ARTICLE INFO

Article history:
Received 29 July 2015
Received in revised form
13 October 2015
Accepted 14 October 2015
Available online 22 October 2015

Keywords:
3D
Rasterization
Voxelization
Scan conversion
Seismic
Finite difference simulation
Parallel computation

ABSTRACT

3D geological underground models are often presented by vector data, such as triangulated networks representing boundaries of geological bodies and geological structures. Since models are to be used for numerical simulations based on the finite difference method, they have to be converted into a representation discretizing the full volume of the model into hexahedral cells. Often the simulations require a high grid resolution and are done using parallel computing. The storage of such a high-resolution raster model would require a large amount of storage space and it is difficult to create such a model using the standard geomodelling packages. Since the raster representation is only required for the calculation, but not for the geometry description, we present an algorithm and concept for rasterizing geological models on the fly for the use in finite difference codes that are parallelized by domain decomposition. As a proof of concept we implemented a rasterizer library and integrated it into seismic simulation software that is run as parallel code on a UNIX cluster using the Message Passing Interface. We can thus run the simulation with realistic and complicated surface-based geological models that are created using 3D geomodelling software, instead of using a simplified representation of the geological subsurface using mathematical functions or geometric primitives. We tested this set-up using an example model that we provide along with the implemented library.

1. Introduction

Simulations are increasingly an integral part of geoscientific investigations. They are, for example, used for predicting fluid flow in hydrogeology (Cheng et al., 2014; Nakajima, 2013) and reservoir planning or for stress-field simulations. In geophysics, numerical simulations reveal information about physical fields at places that cannot easily be accessed for measurement and thus provide a better understanding of the behaviour of the fields and the structure of the subsurface. They have been performed, for example, for electromagnetic modelling (e.g. Wang and Hohmann, 1993; Commer and Newman, 2004) and for developing underground monitoring designs with electromagnetic measurement methods (Börner et al., 2015). Seismic simulations (e.g. Bohlen, 2002; Virieux, 1984, 1986) help to describe and analyse the effects of wave propagation in very complex geological settings and allow virtual seismic sources and receivers to be placed practically everywhere within the model. This is, among other things, a simple and inexpensive way to plan optimized acquisition geometries for real seismic measurements that focus, for example, on the illumination of certain geological structures.

If the simulation is to yield detailed and meaningful results, the modelling domain has to contain all available information about the volume of interest to the best knowledge of geoscientists. This includes petrophysical parameters and a detailed description of the natural and irregular geometry of geological bodies. The geology is usually described by boundary surfaces between these bodies which are represented by irregular triangulated networks (see top left of Fig. 1 for an example). If a set of boundary surfaces confines the volume of a body, this is called a boundary representation, and is described as watertight when boundary surfaces are composed of a conformable triangle mesh without holes and overlaps. In this way we can describe complex geometries with a low amount of memory, since the volume of the body has not been discretized (see top right of Fig. 1).

If numerical simulations are to be performed, systems of partial differential equations have to be solved. This is usually done by way of an approximation for points or cells in the modelling domain. A common method is the finite difference method, which replaces the partial derivatives in these equations with finite difference expressions and most of the work cited above has been done using this method. Finite differences can be easily

* Corresponding author. Fax: +49 3036993100.
E-mail address: bjoern.zehner@bgr.de (B. Zehner).

http://dx.doi.org/10.1016/j.cageo.2015.10.008
0098-3004/

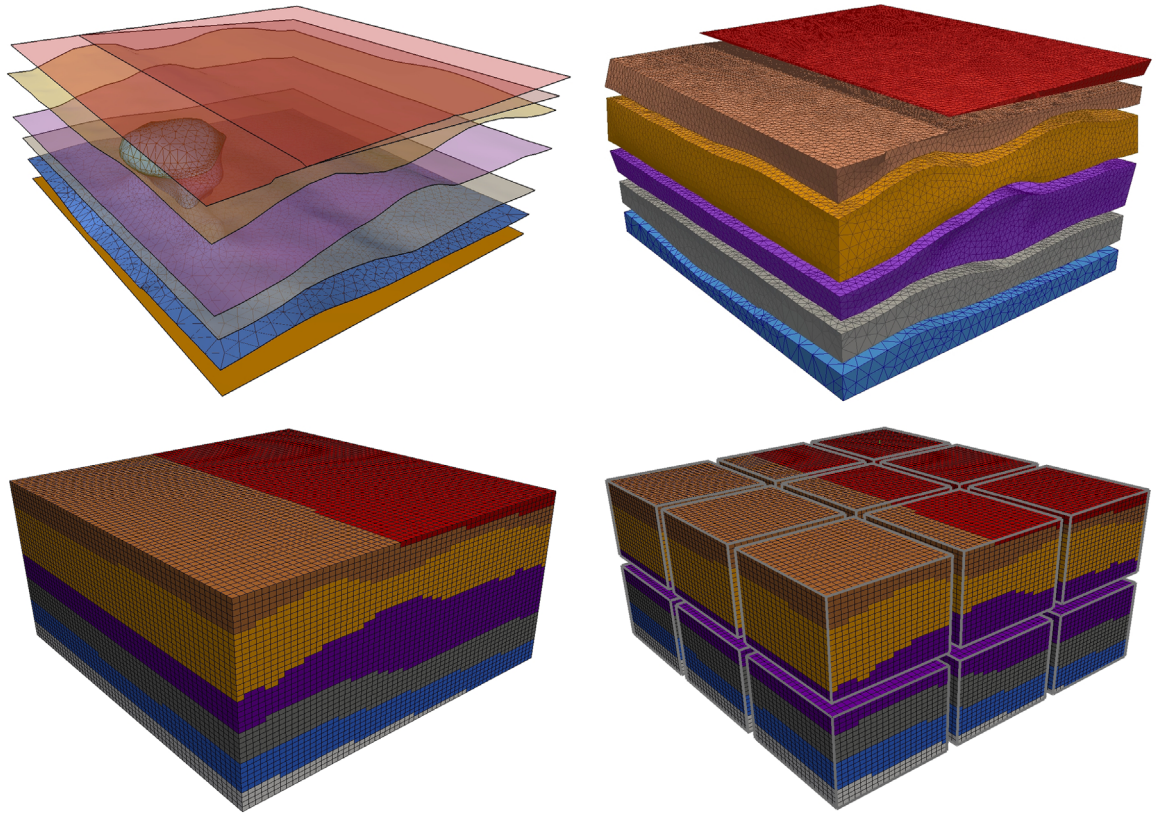

Fig. 1. Four different representations of our example model, a salt dome. Top left: a triangulated representation as is often generated by geologists using standard software. Top right: exploded view of the same model converted to a watertight boundary representation as is needed as input for our rasterizer. Bottom left: rasterized representation using hexahedral cells, as is used by finite difference codes. Bottom right: same as bottom left but for parallelized codes – boxes with grey edges indicate the subdomains for the different processors.

implemented for grids of constant step width with a regular simple topology which can be described by the origin's coordinates, the number of cells and the step width in each dimension. The resulting rectangular hexahedral meshes are used to discretize the domain into cells to which the physical parameters and rock properties are assigned (bottom left of Fig. 1). These types of grid are also advantageous for the use with geostatistics (e.g. Deutsch and Journel, 1998; Yarus and Chambers, 1994; Schaeben, 2014). These grid-based volumetric representations have to be generated from the 3D subsurface models that are represented by triangle meshes.

Many of the common software packages for 3D geological modelling already provide the necessary workflows for generating hexahedral grids. Within Paradigm's software Skua-Gocad (Mallet, 2002, 2008, 2014) and Schlumberger's Petrel software, hexahedral grids could be adapted to the geological peculiarities by distorting the cells geometrically while preserving the topology. Further Skua-Gocad permits the resampling of data onto a regular rectangular grid, a so-called Voxet. Marschallinger et al. (2015) introduced a visual LISP program that makes use of AutoCAD functions to voxelize AutoCAD solid models. Watson et al. (2015) introduced a plugin for the Software GSI3D to generate 3D grids from triangular irregular networks.

If the simulation requires a high spatial resolution, for example, to ensure a certain spatial sampling of the physical fields in order to avoid grid dispersion effects, or because the simulation result is required for this resolution, the grids can become very large. A seismic simulation, for example, that requires a grid with $4000 \times 4000 \times 2000$ cells using 1 Byte per cell as identifier for the stratigraphic unit, would already need approximately 30 GByte of memory or disc space. When physical properties with floating point precision (4 Byte) are involved, 120 GByte of memory are required for each property. Storing seismic P and S wave velocities and the mass density would therefore already require 360 GByte. Apart from these memory requirements, the generation of grids with this size also becomes difficult, if not impossible, using the common commercial software packages for 3D geological modelling.

Simulations that require such a large number of cells are usually implemented as parallelized codes which can utilize several hundred processors per run. This is, for example, the case for the software FD3D which is used for simulating seismic wave propagation and has been developed by the seismic working group at TU Bergakademie Freiberg. So far, due to the parallelization, this software could only be used to run simulations on simple synthetic subsurface models which could be described as simple functions of the x-, y- and z-coordinates. Examples were layer cake models where the layer boundaries could be described as simple planes in space by using trigonometric functions. In order to be able to simulate the seismic wave field for more complex geological 3D models, it would be desirable to use more complex polygonal models that have been generated using standard 3D geomodelling software.

The parallelization of the software FD3D is done by domain decomposition, which breaks down the volume that is to be simulated into several sub-volumes. Each sub-volume is simulated on its own processor (see bottom right of Fig. 1). The exchange of information between the sub-volumes is realized using the Message Passing Interface (MPI). The parallelization solves the problem of calculation on large grids, but not of generating and storing these grids. Therefore, we describe the geology by using a watertight boundary representation where each geological body is

represented by a closed surface made of triangles. Neighbouring bodies have an identical triangulation at their interfaces, so that there are no overlaps or gaps in between. If the geological model consists of different formations with homogeneous elastic properties for each formation, the boundary representation requires much less memory than a highly resolved 3D grid that fulfills the requirements of the simulation. Instead of creating a 3D grid of the whole simulation domain beforehand and reading it on startup, the boundary representation is read by each process and the volumetric model in the form of a hexahedral grid is created and parameterized on the fly for its respective subdomain of the simulation only, using a rasterizer library that we have implemented in C++ and integrated into our seismic simulation software. As an important target, we have implemented the rasterizer library in a general way, so that it could deal with complicated models, such as salt domes, overturned folds and faults. However, as a trade-off it makes high demands regarding the input model.

In this paper we first explain how this C++ library works, how it is implemented, and how it can be used from C++ and ANSI-C programs. Then we explain how the necessary input models can be generated. As an example, we have generated the 3D model of a salt dome shown in Fig. 1 for testing our code and we provide this model along with this paper and the source code for test purposes. Finally the seismic simulation for this model is explained in more detail and some simulation results are given.

2. Rasterizing

Our rasterizer library uses an adapted version of the Ray Shooting Algorithm as it has been, for example, described by Laszlo (1996). A point is classified as being inside a closed polygon if the number of intersections of a ray emanating from this point with the segments of the polygon is an uneven number (see Fig. 2). This approach can be easily extended to the 3D case by evaluating if the ray intersects an even or uneven number of triangles. This requires the triangulation to represent a watertight boundary representation of a volume, where each geological object is described by a polyhedron, such that common boundaries are duplicated. In our sample implementation we shoot the ray in the positive z-direction, as this is ideal for accelerating the algorithm if the geological strata are mainly oriented horizontally. This shooting direction is also assumed in the following discussions.

It is clear that checking if the ray intersects a particular triangle or not cannot be done for each triangle if the algorithm is supposed to run sufficiently fast. Instead we restrict this test to the few triangles for which a hit is likely to occur. We do this with the help of interval trees, which are described in De Berg et al. (1997).

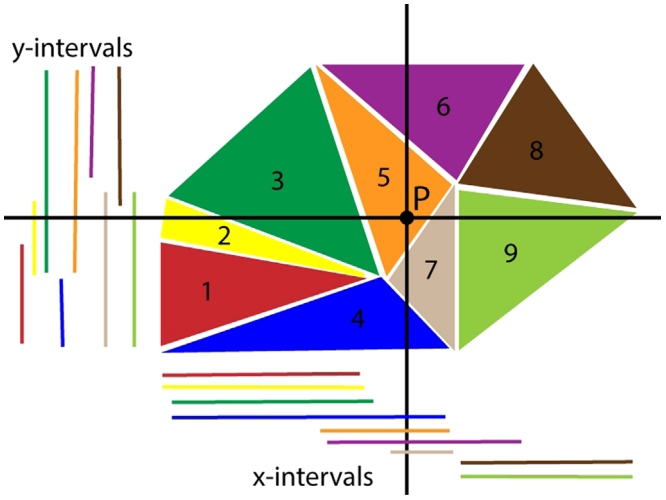

Fig. 3. Search strategy for finding the triangles which potentially contain a point, using two interval trees which store the x- and y-intervals of the triangle's bounding box. If we search for the triangle that contains the point P, we do a stabbing query with the x-coordinate for finding the corresponding x-intervals (vertical black line). In this step the triangles 4–7 are marked. In the second step we do a stabbing query with the y-coordinate of the point (horizontal black line). This would deliver the triangles 2, 3, 5, 7 and 9 but only the triangles 5 and 7 have been marked before. So only the triangles 5 and 7 are potential candidates and must be processed further.

For each triangle, we store the x- and y-intervals of the axis-aligned bounding box in two interval trees (one for each axis), together with a pointer to the triangle. Interval trees constitute an efficient data structure for executing a so-called stabbing query: finding the intervals within a set of intervals that contain a certain value. As a first step, all triangles are marked for which the x-interval of the bounding box contains the x-value of the point. In the second step all triangles are identified where the y-interval of the bounding box contains the y-value of the point and, if these are already marked, they are reported (see Fig. 3). Only for these triangles is the intersection test with the ray performed by calculating the barycentric coordinates of the point in the triangle projected onto the xy-plane. All three barycentric coordinates should be in the range of 0–1 and their sum should be equal to 1, otherwise the point lies outside the triangle. If it is inside, we linearly interpolate the z-value of the triangle at the xy-position of the point. If this z-value is larger than the z-value of the point, we have an intersection with the ray shot from the point in the positive z-direction.

The performance for a stabbing query in an interval tree is $n^* \log(n) + k$ where k is the number of reported intervals (De Berg et al., 1997). In our case the number of k can be quite high for the

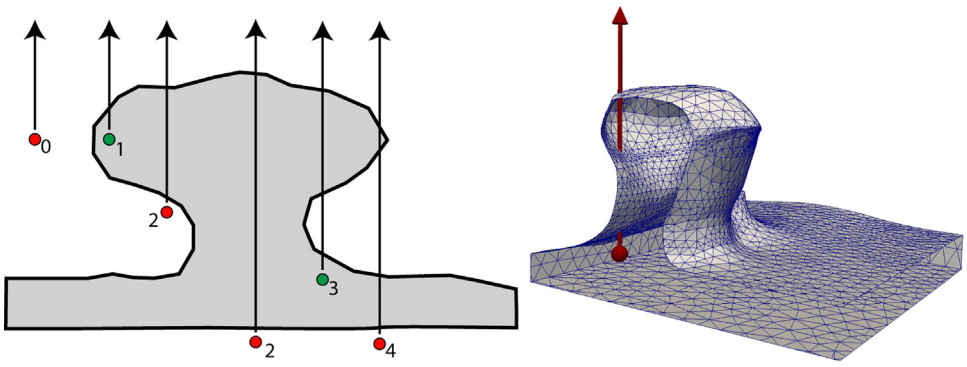

Fig. 2. Illustration of the Ray Shooting Method in the 2D case (left) and the 3D counterpart. A ray is shot in the positive z-direction and the number of intersections with the bounding polygon is counted. If this number is zero or even, the point is outside the polygon (red points), otherwise it is inside (green points). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

stabbing query made in the first step. For this reason we implemented a partitioning of the domain into several subdomains for which the interval trees are generated individually.

3. Implementation

The library consists of the three C++ classes Model3DRasterer, Model3DRastererBRMesh and TwoDIntervallTree which are implemented as templates, the class Model3DRastererDouble, which binds them to double precision as type, and an ANSI-C wrapper that allows for linking the library to ANSI-C programs (see Fig. 4). The main functionality, deciding if a given point lies within a volume, is implemented in Model3DRastererBRMesh. An object of this class is created for each boundary representation of a geological unit. Methods are provided to read the mesh from a file and insert the triangles into search structures. Further a subdomain, projected onto the *xy*-plane, may be defined, so that not all triangles need to be used. If a query is executed to verify if a given point lies within the volume that is defined by the boundary representation, the following steps are executed (it is assumed that the ray is shot in the positive *z*-direction):

1. Is the point above or beside the bounding box of the unit? If yes, return FALSE.
2. Use the interval trees and find all triangles which, due to their bounding box, are potential candidates for intersection.
3. For each triangle:
   (a) Check if the largest *z*-coordinate of the triangle is smaller than the *z*-coordinate of the point. If it is, dismiss the triangle.
   (b) Determine the barycentric coordinates of the point within the triangles system projected onto the *xy*-plane. If any one of them is smaller than zero or their sum is not equal to 1, dismiss the triangle, as the point lies outside of the triangle.
   (c) Determine the *z*-value of the triangle at the point's position, using the barycentric coordinates. If this *z*-value is higher than the point's *z*-value, the triangle is intersected and counted.
4. If the number of intersected triangles is even, return FALSE, otherwise return TRUE.

The class Model3DRasterer is a container class that contains several objects of type Model3DRastererBRMesh and manages them. It provides the interface for the user of the library and executes the query within which volume a certain point lies with help of the objects it manages. If a point is queried, it is initially assumed that the points that are tested sequentially are close to each other and so belong potentially to the same stratigraphic unit. For this reason the Model3DRasterer keeps track of within which unit the last point was lying and tests this unit first. If this test is not successful, it tests the unit with the next larger ID, then the one with the next lower ID and so on. So, for performance reasons, the IDs should be assigned sequentially to the layers for a flat lying layer cake model and sampling on a grid should be done in *xy*-layers.

The C++ library is written as a template which, in our case, uses double precision as type. Many algorithms in computational geometry are not fully stable with this data type, as numerical errors can occur and the tests of whether, for example, two points are the same or if a point lies on a line or in a triangle, can be unreliable. However, the likelihood that a given point is classified incorrectly is very low – we ran a test with 62.5 million cells and it did not occur. Furthermore it would be possible to lower this likelihood substantially by shooting the ray in several directions, for example, additionally in the positive *x*-direction and the positive *y*-direction. Moreover it is possible to perform some additional checks on the generated grid, such as comparing each cell with its neighbours.

One solution for the numerical problem just described would be to use fractions instead of double precision because comparisons would only involve integer numbers in this case, which is numerically stable. The implementation as a C++ template allows the change to this data type to be made. However, optimally this would involve further adaptations of the code. Furthermore the input model would need to be adapted, so that the points only contain coordinates that could be represented by integer numbers or fractions.

For debugging purposes, the library also implements the output of the boundary representation in the file format of the Visualization Toolkit (Schroeder et al., 1996). Furthermore the example code shows how the results could be written to a file in the same format as points. For the visualization of the model and the rasterizing results, the open source software Paraview (www.paraview.org) can be used. For an example see Fig. 5.

The library for rasterizing is integrated into the seismic simulation code with an ANSI-C wrapper. On start up, each process

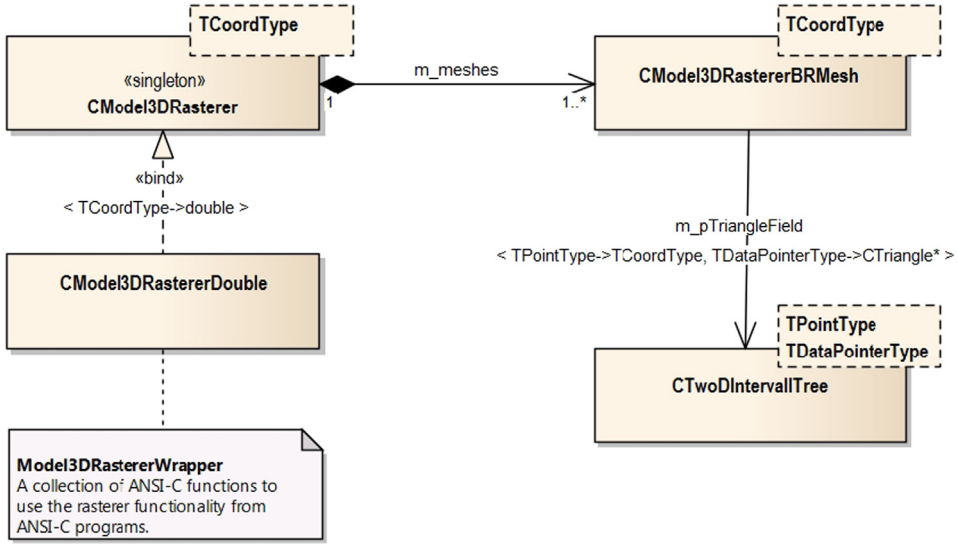

Fig. 4. Class diagram of the rasterizer library showing the main classes. Some further classes, for example defining points and triangles, are defined within these classes' namespaces.

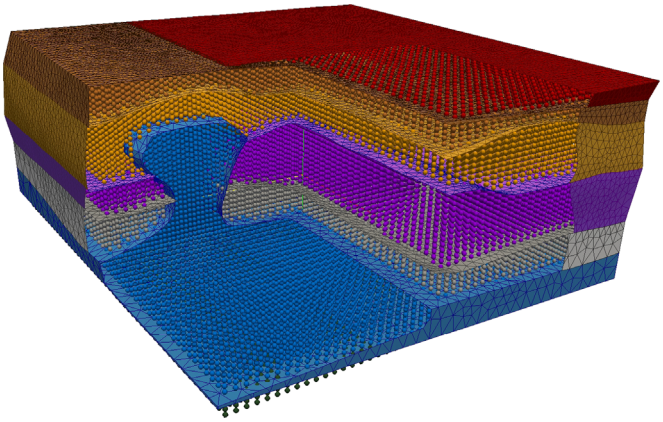

Fig. 5. Rasterized model of the synthetic salt dome. The points represent the centre of the cells.

reads the surface representation of the model and a table with the rock properties for each geological unit. It then determines for which part of the simulation domain it is responsible and allocates the grid for this subdomain. The process then determines for each grid cell to which geological unit the cell belongs and sets the physical attributes to appropriate values.

4. Input model generation and example models

In order to guarantee that even complex and complicated underground models can be discretized in a reliable manner, the library requires as input a boundary representation model of the subsurface where each subdomain or geological body is defined by a watertight triangle mesh as is shown in the top right of Fig. 1. This model must be created from the type of representation in which geological models are often produced by geologists (shown in the top left of Fig. 1). Even if the rasterizer is less demanding regarding the mesh quality of the boundary representation than is the case for the generation of unstructured tetrahedral meshes for finite element simulation, these input models require more effort in terms of construction. Principally, different workflows describing how geological models produced in Skua-Gocad could be prepared for the rasterizer have already been published in Zehner (2011) and Zehner et al. (2015), but they need to be extended slightly. Fig. 6 shows the extension-points in the Skua-Gocad based workflows from Zehner et al. (2015) from where the input models can be generated. See this paper for a more detailed discussion of the original workflows.

The first extension point is the use of the so-called Model3D, which provides a triangulation of the full boundary representation that is conformal at the contacts between different surfaces. The triangulated surfaces that we need as input for the C++ library can be gained from a Model3D using one of Skua-Gocad's standard commands. The second and third extension points start with a TSolid and can be applied, if a geological model has already been provided by a volume discretized with a tetrahedral mesh (in Skua-Gocad called a Solid). This already includes a watertight boundary representation represented by facets of the tetrahedra. The watertight boundary surfaces can be extracted from the boundaries of the Solid's regions using one of Skua-Gocad's standard commands.

The model we use as an example and for test purposes is a synthetic model of a salt dome and represents a salt diapir with its deformed host rocks. Since the diapir has pierced the overlying host rock, the model has a complicated topology with geological bodies containing inner boundaries and it cannot be projected on a plane. We have used the third extension point for its generation. As a first step we have generated a Solid (tetrahedral mesh) and then as a second step the boundary representation.

We have used the salt dome and loaded it into a simple program that uses the rasterizer library in order to carry out a

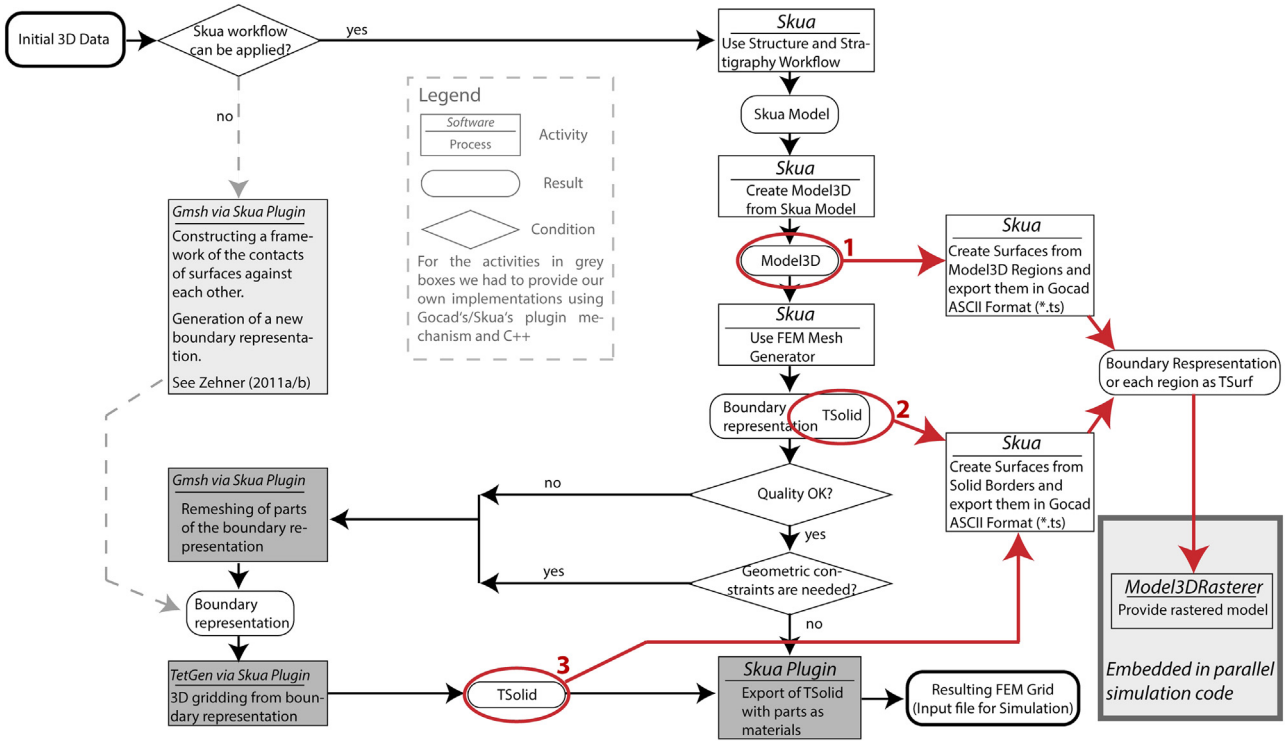

Fig. 6. The workflows for generating the input model for the rasterizer are extensions of the workflows that have been presented in Zehner et al. (2015). The initial workflows were aimed at generating tetrahedral input grids for finite element simulations. The red circles and arrows mark the three extension points referenced in the text. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

performance test. As its performance depends on the input model (number of triangles and configuration of the triangles) we can give only very rough estimates as an example. The synthetic salt dome model consists of 7 units (bodies) whose boundary representations are defined by 110,000 triangles in total. On a workstation with Intel Xeon E5 (3.1 GHz) processor the set-up of the necessary data structures (e.g. the interval trees) needs 0.5 s. Rasterizing this on a grid with $50 \times 50 \times 40$ cells, which means 100,000 cells, as is shown in Fig. 5, requires 1 s. Rasterizing it with $120 \times 120 \times 80$ cells (1,152,000 cells) requires 11 s.

5. Finite difference simulation of seismic wave propagation

The program FD3D uses a finite difference time domain approach to solve the seismic wave equation which, in our case, is based on the velocity–stress formulation (Eqs. (1) and (2)). It is a system of first order partial differential equations that connects the stress tensor components σ_{ij} and the components of the particle velocity $\dot{u}_i$, where the first three equations of the system represent the equation of motion and the following six equations represent the constitutive equations for a linear, elastic and isotropic medium. The wavefield components σ_{ij} and $\dot{u}_i$ are functions of space and time, whereas the material parameters v_P (compressional wave velocity), v_S (shear wave velocity) and ρ (mass density) are time-invariant. The second term on the right hand side of Eq. (1) is a source term with $\vec{f}^S = (f_x^S f_y^S f_z^S)^T$ denoting a force source, such as the force of a drop weight on the ground. $\dot{p}^S$ in Eq. (2) denotes a pressure rate as, for example, caused by an explosive source or an airgun. The components of the particle velocity $\dot{u}_i$ are the physical quantities that are measured by geophones. The pressure p, which can be recorded by hydrophones in marine seismics, can be obtained from the stress components according to $p = -(\sigma_{xx} + \sigma_{yy} + \sigma_{zz})/3$. The solution of Eqs. (1) and (2) together with appropriate initial and boundary conditions yields the full elastic wavefield including body waves (compressional and shear waves) as well as surface waves, for example, Rayleigh waves:

$$\begin{pmatrix} \ddot{u}_x \\ \ddot{u}_y \\ \ddot{u}_z \end{pmatrix} = \frac{1}{\rho} \begin{pmatrix} \partial_x & 0 & 0 & 0 & \partial_z & \partial_y \\ 0 & \partial_y & 0 & \partial_z & 0 & \partial_x \\ 0 & 0 & \partial_z & \partial_y & \partial_x & 0 \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{pmatrix} + \frac{1}{\rho} \begin{pmatrix} f_x^S \\ f_y^S \\ f_z^S \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} \dot{\sigma}_{xx} \\ \dot{\sigma}_{yy} \\ \dot{\sigma}_{zz} \\ \dot{\sigma}_{yz} \\ \dot{\sigma}_{xz} \\ \dot{\sigma}_{xy} \end{pmatrix} = \rho \begin{pmatrix} v_P^2 \partial_x & (v_P^2 - 2v_S^2)\partial_y & (v_P^2 - 2v_S^2)\partial_z \\ (v_P^2 - 2v_S^2)\partial_x & v_P^2 \partial_y & (v_P^2 - 2v_S^2)\partial_z \\ (v_P^2 - 2v_S^2)\partial_x & (v_P^2 - 2v_S^2)\partial_y & v_P^2 \partial_z \\ 0 & v_S^2 \partial_z & v_S^2 \partial_y \\ v_S^2 \partial_z & 0 & v_S^2 \partial_x \\ v_S^2 \partial_y & v_S^2 \partial_x & 0 \end{pmatrix} \cdot \begin{pmatrix} \dot{u}_x \\ \dot{u}_y \\ \dot{u}_z \end{pmatrix} - \begin{pmatrix} \dot{p}^S \\ \dot{p}^S \\ \dot{p}^S \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{2}$$

The finite difference solution requires the discretization of the partial differential equations. For this purpose we define the wavefield components and material parameters on a 3D staggered grid where σ_{xx}, σ_{yy} and σ_{zz} are defined in the centre of the grid cells, σ_{yz}, σ_{xz} and σ_{xy} are defined on the edges and $\dot{u}_x$, $\dot{u}_y$ as well as $\dot{u}_z$ are defined on the faces of the grid cells similar to the 2D staggered grid presented by Virieux (1984, 1986). The staggered grid scheme has the advantage that central finite difference operators can be used to represent the derivatives with respect to space and time without the need for wavefield interpolation. The finite difference approximation of Eqs. (1) and (2) is then solved by an explicit leapfrog time-stepping scheme using second order accurate finite difference operators in space and time. This method is conditionally stable (Courant et al., 1928). The time step width Δt is limited by the Courant–Friedrichs–Lewy criterion, which is given by

$$\Delta t \leq \frac{1}{v_{\max} \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}} \tag{3}$$

where Δx, Δy and Δz are the grid spacings in x-, y- and z-directions. In order to avoid artificial dispersion of the seismic waves propagating over the finite difference grid, it is furthermore necessary to ensure a sufficient sampling of the seismic wavelength. Bohlen and Saenger (2006) propose resolving the shortest wavelength of Rayleigh waves with at least 30 grid points. For an accurate representation of body waves, about 15 grid cells are sufficient. The shortest wavelength can be estimated as follows:

$$\lambda_{\min} = \frac{v_{\min}}{f_{\max}} = \frac{v_{\min}}{3 f_C} \tag{4}$$

assuming that the maximum frequency $f_{\max}$ of the seismic signal is three times its centre frequency f_C.

The seismic wave equation, as it is given by Eqs. (1) and (2), is strictly speaking only valid for homogeneous media. It can be applied to arbitrarily complex media, where the material properties vary from grid cell to grid cell, by appropriate averaging of the material parameters that are needed for the update of the wavefield components on the edges and faces of each grid cell (Moczo et al., 2004). Together with the averaged material parameters, the seismic wave equation fulfills the continuity conditions for the stress and particle velocity components at the interfaces between different media. Free surface boundary conditions are implemented on the edges of the model grid, i.e. the components of the normal stress vanish (e.g. $\sigma_{xx} = \sigma_{xy} = \sigma_{xz} = 0$ for a free surface with its normal pointing into x-direction). In order to simulate unbounded media, we apply Perfectly Matched Layers (PML) in addition to the free surface boundary condition within a frame on the edges of the model grid. The wave equation is modified within the PML frame so that incident waves are attenuated and leave practically no reflections at the model boundary (Drossaert and Giannopoulos, 2007; Komatitsch and Martin, 2007).

In order to compute large 3D models in an acceptable amount of time, our finite difference time domain scheme is parallelized using a decomposition of the spatial finite difference grid into subdomains of cuboidal shape. Each processing element solves the finite difference equation for a single sub-grid. The wavefields from the edges of the neighbouring sub-grids are stored in padding layers and exchanged between the processing elements at every time step. The exchange between the neighbours is realized by the Message Passing Interface (MPI) using persistent, non-blocking MPI routines on an MPI Cartesian topology, i.e. only immediate neighbours communicate with each other. The finite difference simulation yields synthetic seismograms at defined receiver positions and snapshots of the wavefield at certain points of time. The memory required for the output is for most practical cases by far less than the memory required for the material parameters and the wavefield components during the simulation, which is for this reason distributed on several CPUs. The snapshots in particular often cover only a single slice or a limited part of the simulation grid with a coarse sampling.

As a proof of concept and as a test for our system, we have used it to simulate the wave propagation for 6 s in one of our example models, the synthetic salt dome. The material parameters for this model correspond to typical elastic parameters of rocks from a simplified stratigraphic sequence in the North German Basin (Schön, 1983) and are given in Table 1. The simulation was

Table 1
Elastic material properties.

Formation	v_P (m/s)	v_S (m/s)	ρ (kg/m^3)
Cretaceous	2600	1500	2400
Jurassic	2400	1390	2200
Keuper	2820	1630	2190
Muschelkalk	4560	2630	2600
Buntsandstein	4160	2400	2350
Zechstein salt	4500	2600	2090
Rotliegend	4800	2770	2650

performed on a grid consisting of $1100 \times 860 \times 700$ (662,200,000) grid cells and splitting the domain into $5 \times 5 \times 5$ subdomains, so that we could use 125 processors for the simulation, each processor being responsible for approximately 5.3 million cells. The grid spacing for the simulation was $\Delta x = \Delta y = \Delta z = 10$ m and the width of the time step was $\Delta t = 1$ ms. The source excitation function is a Ricker wavelet with a centre frequency of 5 Hz. Running the simulation for a single shot on 125 CPUs (Westmere EP X5670, 2.93 GHz, peak performance 20.25 TFlop/s) took approximately 48 min. A comparison with the numbers given at the end of Section 5 shows that the time for rasterizing the model (approx. 1 min) can be neglected. Fig. 7 illustrates the configuration of the input model with the sources and receivers and the simulation domain.

The simulated wavefield is recorded by 1001 geophones with a spacing of 10 m along the yellow line in Fig. 7. The 100 sources are distributed along the same line with a spacing of 100 m. The time sampling of the seismogram traces is, at 2 ms, twice as long as the time-stepping of the simulation. Fig. 8 shows a snapshot of the simulated wavefield on a vertical section along the receiver line for a single shot, approximately in the middle of this line, after 1.95 s and the corresponding seismogram for the (virtual) geophones. A movie, showing the development of the wavefield through time and the corresponding seismogram, is provided with the electronic version of this paper. The propagation of the wavefronts through the medium can be observed by the help of the snapshots. The waves are refracted and reflected at the layer interfaces. Waves that arrive at the geophone positions appear as events in the synthetic seismogram section. Thus the snapshots reveal information about the wavefield in the simulation where it is not, or is hardly, accessible in real seismic field measurements, which makes it easy to identify recorded events with complicated traveltime curves in the seismogram section and understand their nature.

The synthetic seismic data from the simulation can be used to test different acquisition geometries or different strategies for data processing. They have the advantage that the result can always be compared with the geological model that was used to simulate the data. This provides an inexpensive means to evaluate and optimize the acquisition or the data processing before the actual measurement has taken place. In this context, a simple processing scheme is applied to the traveltime-based seismogram sections from our model in order to derive a seismic depth image, which can be compared with the geological model. The processing of the synthetic data before migration comprises only the application of a time-dependent gain to correct the amplitude decay due to geometrical spreading. Multiples, as for example the multiples between the surface and the top Muschelkalk reflector, have not been removed from the data before Kirchhoff prestack depth migration. Only the energy of the direct wave and refracted waves was muted. The resulting depth image can be loaded into our 3D model. Fig. 9 shows a screenshot of the seismic structural model and the seismic section. The main reflectors in the migrated section, such as the top of the salt structure, the top Muschelkalk and the base of the Zechstein salt, appear at their true position in the model.

6. Discussion and conclusion

We have shown an algorithm and solution for how triangulated surface-based geological models can be used for parallel finite difference simulation on large grids using computer clusters, without having to generate a memory-consuming volumetric representation for the whole domain beforehand. The drawback of our method is that the approach makes heavy demands on the input model: it requires a watertight boundary representation that partitions the volume of interest by surfaces completely confining all geological objects without holes and overlaps. However, we also show how these models can be generated using Paradigm's Skua-Gocad software. The advantage of the method presented here is that it can be applied to complicated geological models, such as faulted, folded and overturned units, duplicated units and bodies with internal boundaries. The rasterizer itself can be applied for sampling onto any type of grid, such as tetrahedral grids, distorted structured grids or on grids represented by cylindrical or spherical coordinates. Our implementation of the rasterizer is

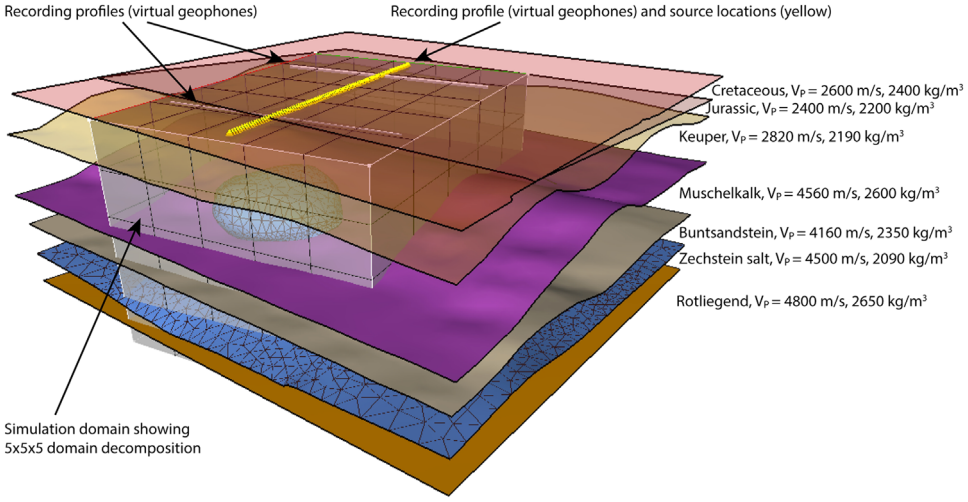

Fig. 7. Configuration of the input for our simulation. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

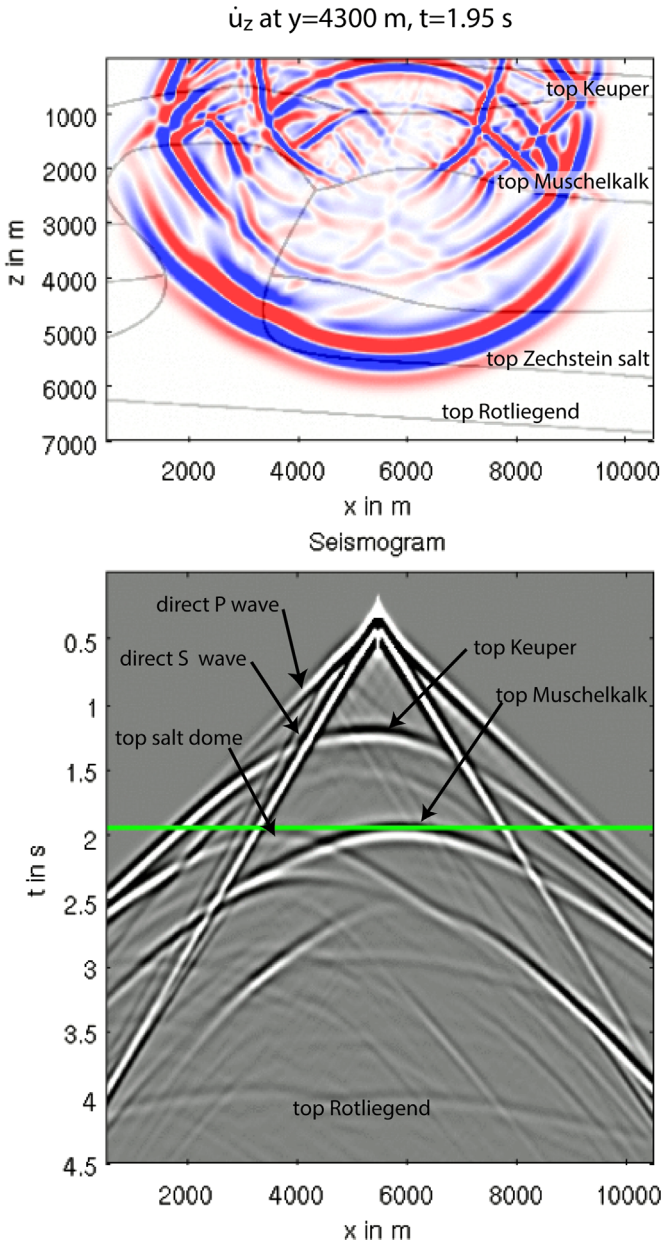

Fig. 8. Top: simulated wavefield for a single shot after 1.95 s extracted along a vertical plane situated along the yellow line in Fig. 7. Bottom: the corresponding seismogram.

relatively lightweight, consisting of only four classes and so it is easy to integrate it into other software and other platforms. Further it can be used from ANSI-C and from C++.

Together with the work described in Zehner et al. (2015), the rasterizer library allows for an integrated approach by using the same 3D geological boundary representation model for generating the input for different simulation techniques. In this way the same model can be used easily first for seismic simulation with finite differences and subsequently for corresponding simulations with finite elements. This will be supported by the flexibility of the rasterizer to adapt to the requirements of different simulation software in terms of cell geometry. For example, FD3D requires that the grids are structured Cartesian grids in order to be able to generate the grids for each subdomain on the fly on the corresponding processor. They cannot be irregularly distorted, but an irregular spacing is possible as it is common practice with finite difference simulations to make the grid more densely spaced in regions of interest. Despite our library initially being specifically tailored to use in FD3D, it is not limited to it and can be generally applied. It could, for example, also be used for writing code that generates input models for non-parallel finite difference simulators, such as SHEMAT (Clauser, 2003), MT3D (Zheng, 1990) or Modflow (Harbaugh, 2005). However, only in the case that the required grids are Cartesian with regular grid spacing do faster solutions exist which are related to scan-conversion of the geometries (see Kaufman and Shimony, 1989 for an overview on algorithms) and can even be further enhanced by using the depth buffer (Karabassi et al., 1999) or modern graphics hardware (e.g. Dong et al., 2004).

Our algorithm is based on ray-tracing and the performance of the rasterizer library could be enhanced by making use of modern graphics hardware (e.g. Purcell et al., 2002). However, in our case the performance of the rasterizer was not an issue and furthermore the individual node-boards of parallel compute systems with hundreds of processors, including the one we use, often do not have programmable graphics boards and so this was not an option for us. Possibly, as also the simulation could be run on graphics hardware (e.g. Rubio et al., 2014), it would then be favourable to reimplement the whole system. Some options that could possibly improve our solution in the future would be to use CGAL's AABB-Tree package (CGAL, 2014) which provides the functionality implemented in our Model3DRastererBRMesh class in a more general and possibly more robust way, but then each time the whole CGAL library would need to be present on the system. Further our solution could be easily enhanced by using OpenMP or threads when multi-core systems are used.

As a next step, we plan to apply the system to a real-world underground model in order to simulate seismic wavefields in a petrothermal reservoir in Schneeberg, Saxony (Germany). A high-

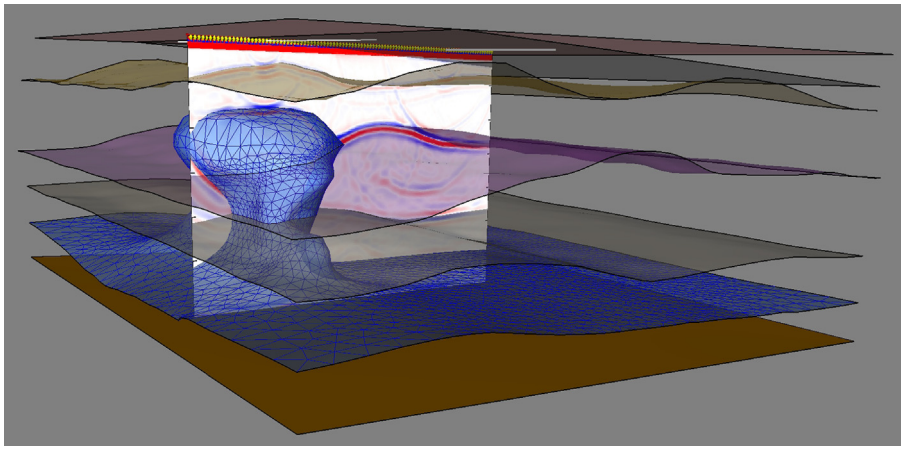

Fig. 9. Simulated and subsequently depth-migrated seismic section blended into the 3D structural model. The image of the layer interfaces Jurassic-top salt, Jurassic–Muschelkalk and the base of the Zechstein salt as main reflectors coincide with the corresponding layer interfaces in the geological model.

resolution 3D seismic survey was carried out at this site in late 2012 with the aim of identifying and characterizing structures that indicate the presence of a petrothermal reservoir (Hloušek et al., 2015). The geological information is gathered in the complex geometry of a 3D model and will be included in seismic simulations. The comparison of the simulated wavefields with the acquired real data will provide additional help with data interpretation, in particular for revealing the nature of geological structures in the target depth for petrothermal usage.

Acknowledgements

We thank Prof. Helmut Schaeben for fruitful discussions during the initial state of this work and for the review of a former version of this paper. Ines Görz and, for half a year in 2013, Björn Zehner were funded by the Ministry of Science and Education of the Federal State of Saxony. Further we thank Alison E. Martin for proofreading the paper and so making it more readable.

Appendix A. Supplementary data

Supplementary data associated with this paper can be found in the online version at http://dx.doi.org/10.1016/j.cageo.2015.10.008.

References

Börner, J.H., Wang, F., Weißflog, J., Bär, M., Görz, I., Spitzer, K., 2015. Multi-method virtual electromagnetic experiments for developing suitable monitoring designs: A fictitious CO2 sequestration Scenario in Northern Germany. Geophys. Prospect., in press. http://dx.doi.org/10.1111/1365-2478.12299.

Bohlen, T., 2002. Parallel 3-D viscoelastic finite-difference seismic modelling. Comput. Geosci. 28 (8), 887–899.

Bohlen, T., Saenger, E.H., 2006. Accuracy of heterogeneous staggered-grid finite-difference modeling of Rayleigh waves. Geophysics 71 (4), T109–T115.

CGAL, 2014. Computational Geometry Algorithms Library. URL ⟨http://www.cgal.org⟩ (last visited March 2015).

Cheng, T., Mo, Z., Shao, J., 2014. Accelerating groundwater flow simulation in MODFLOW using jasmin-based parallel computing. Groundwater 52 (2), 194–205.

Clauser, C., 2003. Numerical Simulation of Reactive Flow in Hot Aquifers—Shemat and Processing Shemat. Springer-Verlag, Berlin, Germany.

Commer, M., Newman, G., 2004. A parallel finite-difference approach for 3D transient electromagnetic modeling with galvanic sources. Geophysics 69, 1192–1202.

Courant, R., Friedrichs, K., Lewy, H., 1928. Über die partiellen Differenzengleichungen der mathematischen Physik. Math. Ann. 100 (1), 32–74.

De Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O., 1997. Computational Geometry—Algorithms and Applications. Springer-Verlag, Berlin, Germany.

Deutsch, C.V., Journel, A.G., 1998. GSLIB—Geostatistical Software Library and User's Guide. Oxford University Press, New York, NY, USA.

Dong, Z., Chen, W., Bao, H., Zhang, H., 2004. Real-time voxelization for complex polygonal models. In: Proceedings of the 12th Pacific Conference on Computer Graphics and Applications, October 2004. IEEE Computer Society, Washington, DC, USA. http://dx.doi.org/10.1109/PCCGA.2004.1348333.

Drossaert, F., Giannopoulos, A., 2007. Complex frequency shifted convolution PML for FDTD modelling of elastic waves. Wave Motion 44, 593–604.

Harbaugh, A.W., 2005. MODFLOW-2005, The U.S. Geological Survey Modular Ground-Water Model—The Ground-Water Flow Process. U.S. Geological Survey Techniques and Methods 6-A16.

Hloušek, F., Hellwig, O., Buske, S., 2015. Three-dimensional focused seismic imaging for geothermal exploration in crystalline rock near Schneeberg, Germany. Geophys. Prospect. 63 (4), 999-1014. http://dx.doi.org/10.1111/1365-2478.12239.

Karabassi, E.-A., Papaioannou, G., Theoharis, T., 1999. A fast depth-buffer-based voxelization algorithm. J. Graph. Tools 4 (4). http://dx.doi.org/10.1080/10867651.1999.10487510.

Kaufman, A., Shimony, E., 1989. 3D scan conversion algorithms for voxel-based graphics. In: I3D—Proceedings of the 1986 Workshop on Interactive 3D Graphics. ACM, New York, NY, USA, pp. 45–75. http://dx.doi.org/10.1145/319120.319126.

Komatitsch, D., Martin, R., 2007. An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation. Geophysics 72 (5), 333–344.

Laszlo, M., 1996. Computational Geometry and Computer Graphics in C++. Prentice Hall, Inc., Upper Saddle River, NJ, USA.

Mallet, J.L., 2002. Geomodeling. Oxford University Press, Inc., New York, NY, USA.

Mallet, J.L., 2008. Numerical Earth Models. EAGE Publications BV, Houten, Netherlands.

Mallet, J.L., 2014. Elements of Mathematical Sedimentary Geology: The GeoChron Model. EAGE Publications BV, Houten, Netherlands.

Marschallinger, R., Jandrisevits, C., Zobl, F., 2015. A visual LISP program for voxelizing AutoCAD solid models. Comput. Geosci. 74, 110–120.

Moczo, P., Kristek, J., Halada, L., 2004. The Finite-Difference Method for Seismologists. An Introduction. Comenius University, Bratislava. ISSN:80-223-2000-5.

Nakajima, K., 2013. Large-scale simulations of 3D groundwater flow using parallel geometric multigrid method. Procedia Comput. Sci. 18, 1265–1274.

Purcell, G.J., Buck, I., Mark, W.R., Hanrahan, P., 2002. Ray tracing on programmable graphics hardware. ACM Trans. Graph. 21 (3), 703–712.

Rubio, F., Hanzich, M., Farrés, A., de la Puente, J., Celaet, J.M., 2014. Finite-difference staggered grids in GPUs for anisotropic elastic wave propagation simulation. Comput. Geosci. 70, 181–189.

Schaeben, H., 2014. A mathematical view of weights-of-evidence, conditional independence, and logistic regression in terms of Markov random fields. Math. Geosci. 46, 691–709.

Schön, J., 1983. Petrophysik. Physikalische Eigenschaften von Gesteinen und Mineralen. Enke Verlag, Stuttgart. ISBN:3-432-92971-4.

Schroeder, W., Martin, K., Lorensen, B., 1996. The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics. Prentice Hall, Inc., Upper Saddle River, NJ, USA.

Virieux, J., 1984. SH-wave propagation in heterogeneous media: velocity–stress finite-difference method. Geophysics 49 (11), 1933–1957.

Virieux, J., 1986. P-SV wave propagation in heterogeneous media: velocity–stress finite-difference method. Geophysics 51 (4), 889–901.

Wang, T., Hohmann, W., 1993. A finite-difference, time-domain solution for three dimensional electromagnetic modeling. Geophysics 58, 797–809.

Watson, C., Richardson, J., Wood, B., Jackson, C., Hughes, A., 2015. Improving geological and process model integration through TIN to 3D grid conversion. Comput. Geosci. 82, 45–54.

Yarus, J.M., Chambers, R.L., 1994. Stochastic Modeling and Geostatistics—Principles, Methods, and Case Studies. American Association of Petroleum Geologists, Tulsa, OK, USA.

Zehner, B., 2011. Constructing geometric models of the subsurface for finite element simulation. In: Conference of the International Association of Mathematical Geosciences (IAMG), September 5–9, 2011. Salzburg, Austria. http://dx.doi.org/10.5242/iamg.2011.0069.

Zehner, B., Börner, J., Görz, I., Spitzer, K., 2015. Workflows for generating tetrahedral meshes for finite element simulations on complex geological structures. Comput. Geosci. 79, 105–117. http://dx.doi.org/10.1016/j.cageo.2015.02.009.

Zheng, C., 1990. MT3D, A Modular Three-Dimensional Transport Model for Simulation of Advection, Dispersion and Chemical Reactions of Contaminants in Groundwater Systems. Report to the U.S. Environmental Protection Agency, 170 pp. On the internet: URL ⟨http://hydro.geo.ua.edu/mt3d/⟩ (last visited November 2014).