The Visual Computer (2019) 35:837–848
https://doi.org/10.1007/s00371-019-01674-x

ORIGINAL ARTICLE

Multiresolution visualization of massive black oil reservoir models

Frederico Abraham[1] · **Waldemar Celes**[1]

Published online: 10 May 2019

Abstract
Recent advances in parallel architectures for numerical simulation of natural black oil reservoirs have allowed the use of very discretized domains. As a consequence, these simulations produce an unprecedented volume of data, which must be visualized in 3D environments for careful analysis and inspection. Conventional scientific visualization techniques are not viable on such large models, creating a demand for the development of scalable visualization solutions. In this paper, we propose a hierarchical multiresolution technique to render massively large black oil reservoir meshes. A new simplification algorithm specialized for such models is presented, which accurately represents boundary surfaces, while keeping the hexahedral mesh with good quality. Original model properties, wireframe and surface normals are mapped onto the simplified meshes through texture mapping. This allows the system to reuse the structure for different simulations that use the same geometry model. The viewer application is designed to guarantee a minimum refresh rate, allocating geometric detail where it is most needed, given the available hardware. Experimental results, considering up to 1.2 billion cell models, demonstrate the effectiveness of the proposed solution.

Keywords Real-time rendering · Reservoir model rendering · Mesh simplification algorithm · Multiresolution rendering · Massive model visualization

1 Introduction

Black oil reservoirs are formed through the accumulation of hydrocarbons in sedimentary rocks [6]. After wells are drilled along the reservoir field, oil and gas are extracted using natural or induced underground pressures, pumping them to the surface using a pipeline network. The oil industry uses numerical simulators to predict the fluid flow throughout the production time of oil and gas fields. Reservoir maximum economic return is pursued through many simulations of different well arrangements and configurations. Simulations are also used later in the production process, using past production data to adjust model parameters. This process reduces uncertainties by improving future predictions.

In order to enable the analysis of such models, which can be composed of very large data sets, a number of scientific visualization techniques are applied. One of the greatest challenges of black oil reservoir visualization nowadays is the ability to handle very large models. The increasing availability of computer power has allowed engineers to drastically improve simulation accuracy by modeling very discretized domains. Models are now composed of tens of millions of cells, and advances in parallel simulations have allowed the simulation of billion-cell reservoirs [11,13]. Such model sizes are unprecedented, requiring the construction of scalable visualization systems in order to inspect these models at interactive frame rates.

Most approaches to handle massive models either employ distributed visualization or level-of-detail techniques. Distributed visualization handles the issue by using more computing power, but this introduces challenges on hardware and software complexities [1]. Scalability is also a concern, as with any distributed algorithm. On the other hand, level-of-detail techniques can do a better job at spending the available computing power where it is needed. This is done by simplifying complex geometry without perceptible quality loss

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00371-019-01674-x) contains supplementary material, which is available to authorized users.

✉ Frederico Abraham
devotion97@gmail.com; fabraham@tecgraf.puc-rio.br

Waldemar Celes
celes@tecgraf.puc-rio.br

[1] Computer Science Department, Tecgraf/PUC-Rio Institute, Pontifical Catholic University of Rio de Janeiro, Rua Marquês de São Vicente 225, Rio de Janeiro, Brazil

in the visualization. Another advantage of the multiresolution approach is its simpler hardware setup, offering quality visualizations on off-the-shelf graphics hardware.

To the best of our knowledge, no simplification scheme was proposed for black oil reservoir meshes. Its particularities require a specialized solution. In this paper, we present a new multiresolution technique to render large-scale reservoir meshes. A new simplification algorithm specialized for such models is presented, which exploits the layered nature of their geometry. A set of cell column collapse operators was devised to simplify the reservoir mesh, aiming to maintain hexahedrons with good shapes and boundary surfaces with accurate geometry representation. The algorithm receives and produces meshes composed of hexahedral cells only, allowing the use of most visualization algorithms devised for regular sized reservoir models. A multiresolution structure was used for out-of-core partition, construction and visualization of such meshes.

Our proposal includes the use of global *topological coordinates*, which are per-vertex $[i, j, k]$ coordinates. These coordinates are added as original vertex attributes and interpolated throughout the simplification process. In visualization time, original model properties and surface normals associated with each cell $[i, j, k]$ are stored in packed 3D textures. The interpolated topological coordinates allow us to map the original model wireframe, properties and surface normals onto the simplified meshes. Decoupling geometry simplification from property mapping effectively allows the reuse of the built multiresolution structure for different simulations that use the same geometry model.

Our multiresolution visualization system has proven to be scalable in relation to state-of-the-art simulators, currently allowing the visualization of a reservoir model with 1.2 billion cells on off-the-shelf computer hardware.

The remainder of this paper is organized as follows: in the next section, related works on mesh simplification, multiresolution hierarchies and texture mapping are presented. Section 3 briefly describes black oil reservoir models and enumerates multiresolution structure requirements for such models. Section 4 presents our proposed reservoir mesh simplification algorithm and multiresolution hierarchy. The techniques proposed to map the original wireframe, properties and normals onto the simplified mesh are presented in Sect. 5. Section 6 presents our out-of-core viewer application, including data layout and view-dependent visualization algorithm. Experimental results are detailed in Sect. 7, demonstrating the effectiveness and efficiency of our solution. Finally, in Sect. 8, concluding remarks are presented and future work is discussed.

2 Related work

Sousa et al. [16] present a map of the complex processes and tasks involved in dealing with geoscience and reservoir models. They outline the major problems and challenges that motivate research on interactive visual computing systems for such models. Their work indicates that one of the major concerns in the development of such applications is handling the exponential increase in data volume. In this work, we focus on visualizing massive black oil reservoir models, characterized by a complex set of hexahedral cells. No previous proposal was presented for managing scalability of such models. However, there is a vast literature on mesh simplification and multiresolution schemes for surface visualization. As we explore the layered composition of reservoir models, metrics and simplification operators for surfaces can be adapted for our purpose.

Garland and Heckbert [8] have presented a surface simplification algorithm based on iterations of vertex pair contractions. The work computes a good approximation of the error committed at each contraction by using a quadric metrics. A symmetric 4×4 quadric matrix Q is built to provide the sum of the squared distances of any vertex to a set of planes, in the case, the planes of the original mesh faces that contain the vertex. Such matrices are computed and stored at each original mesh vertex. Quadric matrices can be added: given a contraction $(v1, v2) \rightarrow \bar{v}$, the matrix $\bar{Q} = Q_1 + Q_2$ is a good approximation for the quadric matrix associated with $\bar{v}$. The error at any vertex v can be computed as $v^{\mathrm{T}} Q\, v$. Quadric matrices are a very compact error representation, requiring the storage of only 10 floating points. We employ their scheme to evaluate the error in reservoir layer elevations and external faces when simplifying the model.

Wu and Kobbelt [21] presented a new mesh decimation framework based on the probabilistic optimization technique of multiple-choice algorithms. The idea is to replace the commonly used priority queue of iterative greedy algorithms. By selecting the best among a small number of randomly selected local operations, many heavy operations are avoided, such as the construction of the priority queue on initialization, the processing of the geometrical change for all entities involved in a local operation, and the subsequent update of their positions in the priority queue. This simple modification has achieved simplification times 2.5 times smaller than the greedy version of the algorithm, while producing meshes with the same quality.

Daniels et al. [7] proposed a quad mesh simplification algorithm that treats the difficult problem of keeping quadrilateral connectivity throughout the simplification. The method uses unit operations applied to the dual mesh representation, improving the mesh structure and maintaining the topological *genus*. The proposal includes an extension to the quadric error metrics for quadrilateral meshes, also prioritiz-

ing collapses that restore vertex valences to the ideal number of four and collapses that create square elements. Tarini et al. [19] presented an incremental method for quadrilateral mesh simplification, with an objective function that allows the progressive generation of a mesh with convex, right-angled and equally sided quads. Their set of local operations is very simple yet very powerful. Another important feature is that it only uses quad elements, not requiring temporary triangles. These quadrilateral mesh simplification strategies are used as inspiration to preserve good hexahedral element shapes, as we adopt a simplification scheme based on column collapses.

Cignoni et al. [5] introduced Adaptive TetraPuzzles, a technique for out-of-core construction and visualization of very large surface models. They utilize a regular conformal hierarchy of tetrahedra to spatially partition the model. Each tetrahedron cell contains a precomputed simplification of the original model, which is stored using a format optimized for rendering. The multiresolution structure is constructed during a fine-to-coarse simplification of the surface contained in diamonds, which are sets of tetrahedral cells sharing their longest edge. Given appropriate boundary constraints during the simplification phase, all conforming subdivisions of the tetrahedron hierarchy result in correctly matching surface patches. In visualization time, the hierarchy is traversed top-to-bottom given the screen and camera parameters. A tetrahedron node is loaded and its associate mesh is rendered if its error metrics is accepted when projected to the screen; otherwise, traversal continues to its children. To guide our column-based simplification algorithm, we use a 2D version of such scheme, creating a conformal hierarchy of triangles projected on the horizontal plane.

Celes and Abraham have proposed the wireframe texture [4] for mesh wireframe rendering based on texture mapping. We extend that approach for 3D meshes and also use it for normal and property mapping.

3 Black oil reservoir model

The black oil reservoir model is created by discretizing the reservoir domain into a tridimensional topological grid with $ni \times nj \times nk$ hexahedral cells. Each cell with topological coordinates $[i, j, k]$ has the cells $[i+1, j, k]$, $[i-1, j, k]$, $[i, j+1, k]$, $[i, j-1, k]$, $[i, j, k+1]$ and $[i, j, k-1]$ as its topological neighbors. The geometry associated with this topological grid is usually irregular, as shown in Fig. 1. Geological *faults* result in discontinuities in elevation, causing topological neighbor cells to not share faces. Some cells might be set as *inactive*, possibly resulting in irregular and/or disconnected cell groups. All cells with the same k coordinate are said to be in the same *layer*; the top faces of the cells in a layer resemble an irregular terrain with possible discontinuities.

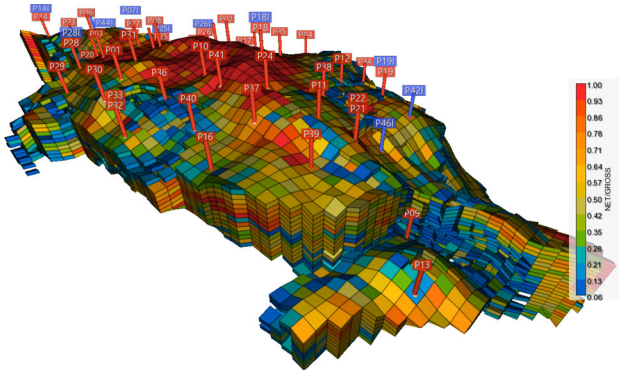


Fig. 1 Reservoir model with irregular geometry

The reservoir is characterized by assigning geophysical and geological properties to its cells. Given a well arrangement and its associated production plan, the simulator computes oil (and gas) flows and other properties, such as bottom-hole pressures, based on its numerical model. The simulation also outputs physical properties associated with each grid cell for each simulation time step, such as oil, gas and water saturations, pressures and temperatures. In this work, the focus is on rendering very large reservoir models with their cell properties. It is also important to allow the visualization of the mesh wireframe, since it helps the specialist to judge the quality of the numerical simulation.

When designing our multiresolution scheme for reservoir models, we set ourselves the challenge to preserve, as far as possible, some characteristics of the original model:

- The structure of layers and columns of cells should be preserved;
- The generated meshes should be composed of hexahedral cells only;
- Hexahedral cells should be well shaped, with internal angles close to 90°;
- A good approximation of the model active cell boundary must be pursued, including geological faults;
- Mesh wireframe and cell properties should be visualized with minimal distortions;
- Simplification should be decoupled from properties, to allow reuse on different simulations of the same geometry model.

In special, preserving the topological structure of layers and columns using only hexahedral cells allows easy adaptation of traditional visualization algorithms used for reservoir model inspection, such as positioning arbitrary cutting planes and filtering regions of interest.

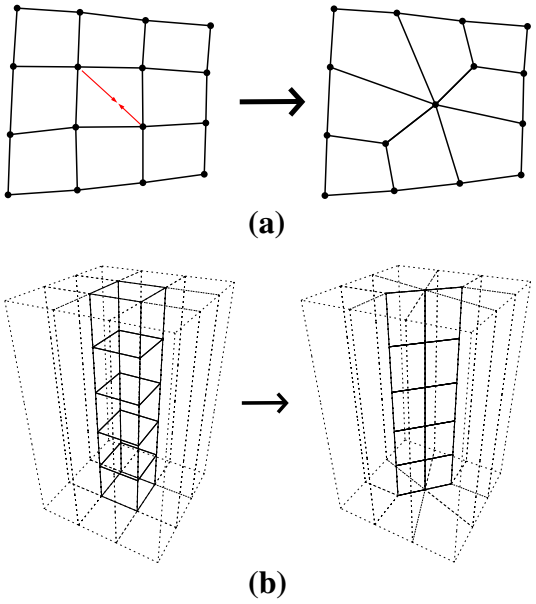


Fig. 2 **a** Element close operator for quad meshes. **b** Element column collapse for hexahedral meshes

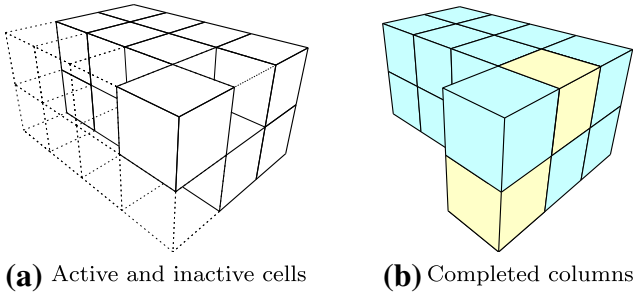


Fig. 3 **a** Original model. **b** Selected columns with active cells marked in blue

4 Multiresolution hierarchy construction

In this section, we describe our proposal to build multiresolution representations of reservoir models. First, we describe a column-based simplification algorithm to reduce the number of cells of a given sub-mesh; then, we present how this simplification algorithm is used to create the multiresolution hierarchy; finally, we describe how to extract information to feed traditional iso-contouring visualization algorithms.

4.1 Reservoir hexahedral mesh simplification

Blacker and Stephenson [3] defined the *element close* operator for quadrilateral meshes, which merges two opposite nodes of a quadrilateral, reducing the mesh quad count by one, as shown in Fig. 2a. Staten et al. [18] considered the extension of this operator for hexahedral meshes, which removes element *columns*, defined as a sequence of hexahedral elements adjacent to one another through their top and bottom faces, as illustrated in Fig. 2b.

Our proposed algorithm for mesh simplification receives a target number of cells. The input reservoir mesh has an active flag associated with each cell. We also set the topological coordinates as vertex attributes, which are carried out during simplification by linear interpolation. The algorithm always performs the collapse of element columns, defined, in the case of reservoir models, as the set of cells with the same i and j topological coordinates. Only reservoir columns containing at least one active cell are considered. Some columns are incomplete, containing both active and inactive cells, as pictured in Fig. 3a. We propose completing all columns considered in the simplification, including their inactive cells as part of the model, as illustrated in Fig. 3b. This guarantees all collapses are performed on columns with nk cells, simplifying many algorithm design decisions. Separated layer visualization is also naturally supported.

In order to keep a good measure of geometric errors throughout the mesh simplification, we associate a quadric matrix with each original mesh vertex, as in [8]. Layer separation support requires the consideration of all top and bottom faces of active cells (faces of *type 1* in Fig. 4) when computing quadric matrices, so their planes are accumulated in their vertex quadric matrices. A side face only contributes to its vertex quadrics if it belongs to an active cell and is *external*. Displacements from internal side faces do not impose error on the external reservoir surface. Three types of external side faces exist (see Fig. 4):

- Faces that represent the boundary between active and inactive cells (*type* 2);
- Faces modeling a geological fault, which are not shared by two cells that are adjacent on the topological grid (*type* 3);
- Faces that compose the external model boundary—faces at the limits of the topological grid (*type* 4).

The planes of these faces are accumulated at each of their vertex quadrics with a weight set to a very large value,[1] guiding our simplification process to respect model boundaries as much as possible.

A column collapse joins vertices through all model layers. The quadric associated with each new vertex is computed as the sum of the quadrics associated with the vertices it replaces, as in [8]. The cost associated with each column collapse is computed as the *maximum error* among the affected vertices.

Our simplification algorithm relies on a set of five collapse operators, illustrated with 2D top views in Figs. 5, 6 and 7.

[1] We used a weight of 10,000 in our experiments.

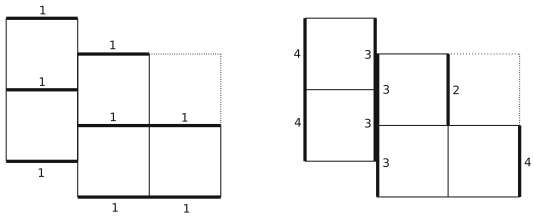


Fig. 4 Side view of a reservoir mesh showing the considered face types when computing quadric matrices. The rightmost cell at the top is inactive. Considered face types: (1) top and bottom faces; (2) side faces shared between active and inactive cells; (3) side faces on geological faults; (4) side faces on the external model boundaries

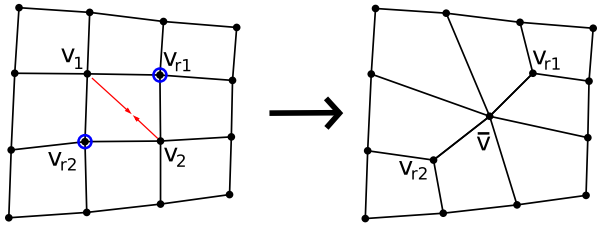
The first operator type (Fig. 5a) collapses an internal mesh column: the quadric of each new vertex $\bar{v}$ is given by $\bar{Q} = Q_1 + Q_2$. We only consider one direction of collapse, merging v_1 and v_2. This direction is chosen by considering the vertex *lateral valence*, which is the number of side faces sharing the vertex, corresponding to the number of incident edges in the top view illustrations presented. The idea is to keep the mesh with low lateral valences, producing meshes with good quality. We then choose the direction that contains the vertex with the lowest valence between the four vertices. If there is a tie between the two directions, the one with the smallest new vertex valence ($\text{val}(\bar{v}) = \text{val}(v_1) + \text{val}(v_2) - 2$) is chosen, and a random direction is chosen if the tie persists. The lateral valences of vertices in the other direction, v_{r1} and v_{r2}, are reduced by one after the collapse.

The second collapse operator type (Fig. 5b) is applied on columns where top and bottom faces have one or two vertices at the model side boundary. These collapses are evaluated in both directions in order to preserve model boundaries as much as possible. The direction with the smallest associated error is chosen. Note that the lateral valences of v_{r1} and v_{r2} are again reduced by one with this collapse.

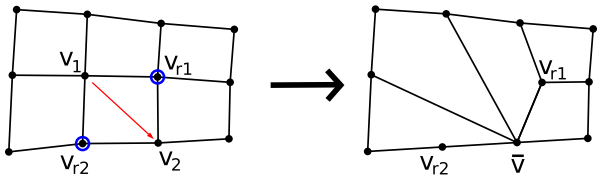
The third collapse operator type (Fig. 5c) is our proposal to remove columns with two consecutive boundary side faces, including corner columns. The quadric of the new vertex $\bar{v}$, joining vertices v_1, v_2 and v_3, is given by $\bar{Q} = Q_1 + Q_2 + Q_3$. The position of the new vertex $\bar{v}$ is set to be the position with the smallest quadric error among v_1, v_2 and v_3.

We also propose a fourth collapse operator type (Fig. 6), acting on columns with one internal side face and three boundary side faces, with all vertices on the model boundary. This configuration type appears after applying the second or third collapse operator types. Two collapse options are given by choosing which vertex of the internal side face will remain in the mesh (v_r). The three other vertices (v_1, v_2 and v_3) are joined and placed at the position with smallest quadric error among the three vertex positions.

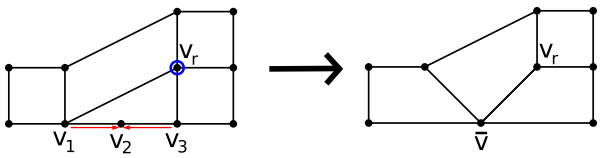
The vertices remaining in the mesh after each collapse were highlighted in blue in all figures and have their lateral valences reduced by one. Whenever the lateral valence of a



(a) Collapse operator for internal columns (top view). The vertices remaining in the mesh are highlighted in blue.



(b) Collapse operator for columns with one or two vertices at the boundary (top view). The vertices remaining in the mesh are highlighted in blue.



(c) Proposed collapse operator for columns with three vertices at the boundary (top view). The vertex remaining in the mesh is highlighted in blue.

Fig. 5 The first three column collapse operators

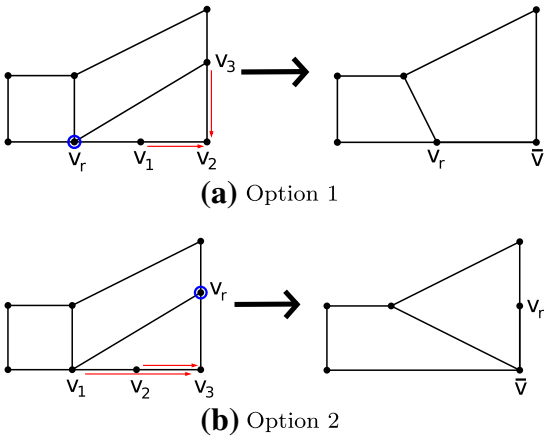


Fig. 6 Proposed collapse operator for columns with four boundary vertices (top view): **a** first collapse option; **b** second collapse option. The vertex remaining in the mesh is highlighted in blue

non-boundary vertex is reduced to 2, a configuration known as a *doublet* is formed, where two adjacent columns share two consecutive side faces, as illustrated in Fig. 7. Vertex v_d contains two internal angles which sum 360°, so it is shared by at least one element with an undesired shape. As in previous works [7,19], doublets are detected and removed, as soon as they are formed, by moving vertex v_d to vertex v_r. The

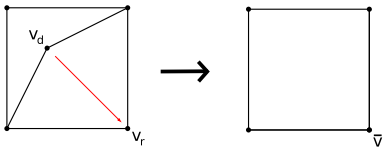


Fig. 7 Doublet collapse, top view

quadric matrix of the new vertex $\bar{v}$ is given by $\bar{Q} = Q_d + Q_r$. This operation can form new doublets: vertices adjacent to v_d also have their lateral valences reduced by one. Therefore, the process is repeated until no doublets are detected.

Several works on finite element mesh generation have stated the importance of smoothing [14,17], in order to generate quad meshes with good quality, e.g., angles as close to right angles as possible. The whole mesh should ideally be smoothed after each column collapse, but that would be extremely costly. We propose local smoothing at each step of the simplification process. At each collapse, we perform a local Laplacian smoothing for each layer (top and bottom vertices). All created vertices and the vertices that remain in the mesh participate in the smoothing process, as long as they are not on the side boundary. New and smoothed vertex attributes are also computed in the Laplacian smoothing process. After determining the x and y coordinates of each vertex, we minimize quadric errors in the z direction using the partial derivative of the quadric error in the z direction. The quadric error expression is given by $\Delta(v = \{x, y, z\}) = v^T Q\, v = q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 + 2q_{23}yz + 2q_{24}y + q_{33}z^2 + 2q_{34}z + q_{44}$, where q_{ij} are elements of the quadric matrix Q. The error partial derivative in the z direction is $\delta\Delta/\delta z = 2q_{13}x + 2q_{23}y + 2q_{33}z + 2q_{34}$. Given that x and y are fixed, the z coordinate with minimum error can be found where the partial derivative is zero: $z = \frac{-q_{13}x - q23y - q_{34}}{q_{33}}$, with $q_{33} \neq 0$. After computing the vertex coordinates of the top and bottom faces of a given column, we check for face fold-overs. If any face fold-over is detected, the column collapse is deemed invalid for the given simplification step.

Our simplification algorithm operates a series of column collapses. The next column to be collapsed is chosen based on the committed quadric error, as in previous works [7,8,21]. As in the work of Wu and Kobbelt [21], the simplification algorithm randomly selects eight column collapses and performs the one with the smallest cost. (In fact, we have experimentally attested that maintaining a priority queue is costlier.)

The following steps are made in each algorithm iteration:

- Choose eight column collapse candidates:
  - Obtain a random column;
  - Select the collapse operator type based on local topology;
  - Compute potential new vertices' positions and attributes;
  - Discard collapse if fold-overs are detected;
  - Compute the quadric error to be committed if the column is collapsed.
- Perform the collapse with smallest quadric error among all candidates;
- Detect and remove any generated doublets;
- Stop simplification if target mesh size is reached.

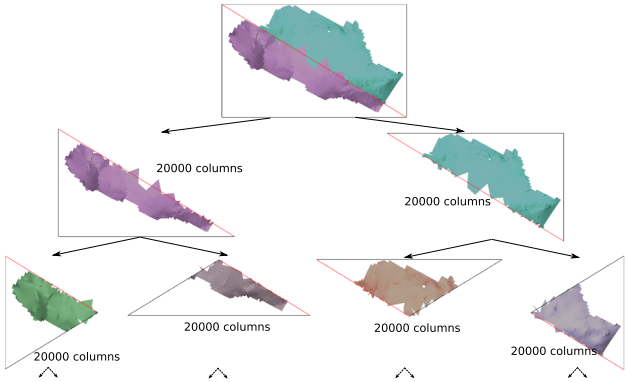


Fig. 8 Used multiresolution hierarchy: triangular regions with associated meshes

4.2 Multiresolution hierarchy

Our multiresolution structure is a 2D version of the Adaptive TetraPuzzles structure by Cignoni et al. [5]. The 2D model bounding box (x and y ranges) is first divided into two triangular regions sharing the box diagonal, forming the two roots of the hierarchy. A recursive subdivision of each triangular region is then initiated. If a region contains more model cell columns than a given threshold C_{max}, it is divided into two triangular regions by the longest edge bisection. Otherwise, the region becomes a hierarchy leaf. In this case, we form a hexahedral mesh with the leaf cell columns, compute quadric matrices and attributes for each vertex and save the result to disk. This process is illustrated in Fig. 8.

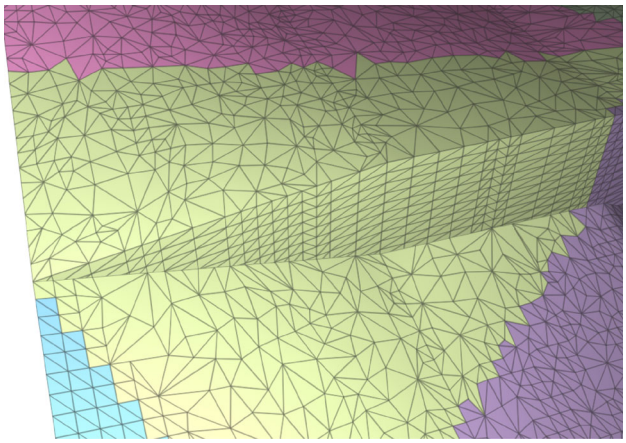
A bottom-up simplification procedure is then initiated, centered on the hierarchy *diamonds* [20], defined as the pairs of triangular regions sharing their longest edge. The simplification on each internal hierarchy diamond is done as follows: first, the meshes associated with the children of the two triangular regions of the diamond are merged. The columns of this resultant mesh that intersect the diamond external boundary are locked. The mesh is simplified until each triangular region holds C_{max} cell columns. The meshes of the two triangular regions are then saved to disk, including associated quadrics. This process continues until all levels are generated, including the top level, composed only by the root diamond.

At visualization time, a conformal 2D space subdivision is obtained by consistently subdividing the triangular regions forming each diamond. Correctly matching mesh patches are intrinsically guaranteed by this scheme. Additionally, this scheme allows diamonds of each level to be simplified in parallel.

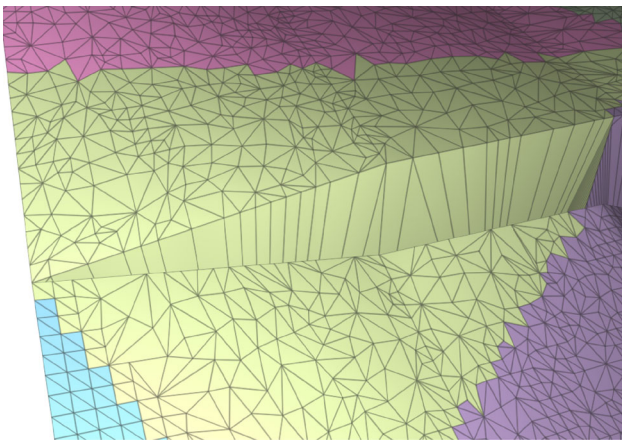
4.3 Simplification of side faces

After the whole multiresolution hexahedral mesh hierarchy has been computed, it is time to prepare it for the desired visualization techniques. Our viewer application implements traditional iso-contouring. To support it, the quadrilateral meshes representing the external faces of each hexahedron mesh are extracted in a preprocessing stage. Three meshes are formed: the top face surface, the bottom face surface and the side face surface.

The side face surface can be further simplified for the visualization of the whole model (without layer separation). First, we recover a simplification error limit, equal to the maximum quadric error committed in the simplified hexahedral mesh vertices. Two constraints must be respected to prevent mesh cracks. A vertex must be locked: (1) if it is part of any external top or bottom face; (2) if it lies outside the associated triangular region. The quad mesh is converted into a triangle mesh and simplified until the maximum error has been reached. A simple iterative edge collapse operator is employed, prioritizing collapses with the smallest quadric error. The final mesh triangle count can be greatly reduced, specially if we are simplifying reservoirs with many layers. Figure 9 illustrates the reduction in triangle mesh size of side surfaces resulting from this procedure.



(a) Mesh without side faces simplification



(b) Mesh with side faces simplification

Fig. 9 Simplification of the triangle surface mesh generated by side face extraction: **a** mesh without side face surface simplification; **b** mesh with side face surface simplification. The meshes associated with each triangular region have been painted with different colors

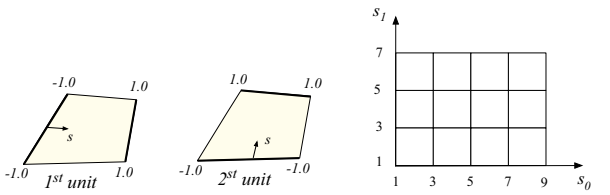
5 Property and grid mapping

One big challenge of multiresolution structures for scientific visualization is the handling of the original model properties. In the case of reservoir models, each simulation can contain many scalar and vector fields associated with each mesh cell, and each field can vary over the simulation time steps. Another challenge is to visualize the original mesh wireframe, which can be useful to reveal structural information about the underlying reservoir domain modeling. We propose viewing wireframe and model properties on top of the simplified meshes by employing texture mapping.

5.1 Wireframe rendering

This work relies on the work by Celes and Abraham [4] to map the original grid wireframe onto the simplified meshes. Their work introduces the *wireframe texture*, a carefully built 1D texture that allows a mesh wireframe to be rendered in a single pass along with polygon rasterization. A quadrilateral can be rendered with wireframe by enabling the texture in two texture units, one for each quadrilateral direction, with texture coordinates set as -1 and $+1$, as shown in Fig. 10a [4]. In the case of structured quad meshes, it is possible to specify texture coordinates based on a global topological coordinate system, using odd consecutive numbers along the two directions, as illustrated in Fig. 10b [4].

Let us consider a 2D reservoir mesh, where each cell is represented by a quad with indices i, j in a $ni \times nj$ topological grid. It is simple to attribute wireframe texture coordinates to the vertices of this mesh: assuming all indices start in 0, the following global texture coordinates are attributed to the vertices of cell $[i, j]$: $[2i+1, 2j+1]$, $[2(i+1)+1, 2j+1]$, $[2(i+1)+1, 2(j+1)+1]$ and $[2i+1, 2(j+1)+1]$. We denominate coordinates i, j in this scheme as the *topological coordinates* of each vertex. It is simple to verify that this coordinate attribution scheme works well even with geological faults—there



(a) Texture coordinates for quadrilateral wireframe rendering **(b)** Global texture coordinate system

Fig. 10 Texture coordinate attribution for quadrilateral mesh wireframe rendering

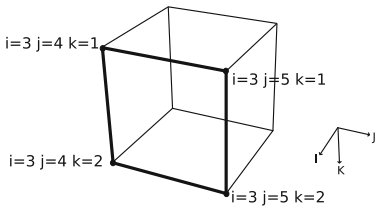


Fig. 11 Grid mapping in 3D: i coordinates do not vary along the face

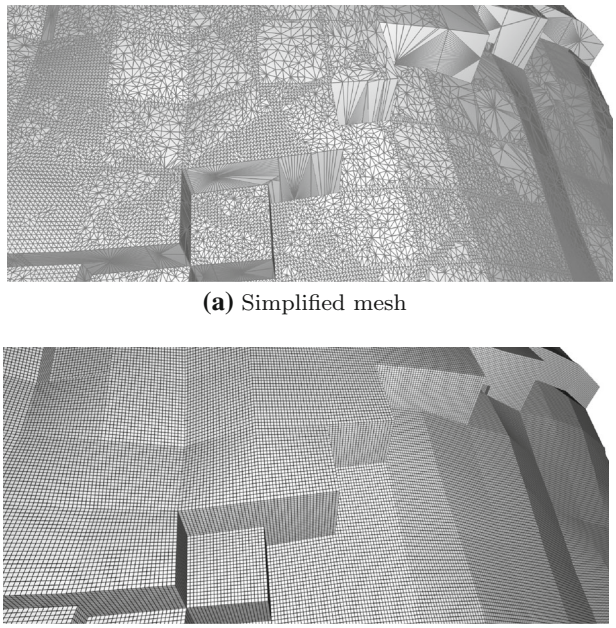
will be different vertices in the same topological position, which is not an issue for their primitives rasterization.

The extension of this scheme to our semi-structured hexahedral mesh is made by including the k coordinate. Since we render the reservoir external faces, one extra step must be made. Only two of the three topological coordinates must be used to map the wireframe texture on each quadrilateral. The excluded coordinate is the one with smaller variation along the primitive. We employ a geometry shader to identify and exclude the coordinate, computing a sort of "topological normal vector," excluding the coordinate in the vector direction. In the illustration in Fig. 11, the j and k coordinates will be used to map the wireframe texture, since the topological normal vector is in the i direction.

These global topological coordinates are included as vertex attributes in our hexahedron mesh simplification algorithm. They are first computed when extracting the original meshes at leaf triangular regions. Then, each collapse interpolates the coordinates linearly whenever a vertex is created or repositioned. Interpolation is also done when simplifying side faces. This allows us to render the original wireframe decoupled from the used hexahedral mesh, working well in any hierarchy level. Figure 12 presents the visual results: Fig. 12a shows a simplified mesh set; Fig. 12b shows the original grid mapped over the simplified meshes. The obtained image is practically identical to the one generated with the original mesh, other than minor mapping distortions.

5.2 Property mapping

We also propose rendering original model properties based on texture mapping, setting appropriate texture coordinates



(a) Simplified mesh

(b) Wireframe mapped over the simplified mesh

Fig. 12 Wireframe rendering decoupled from geometry support

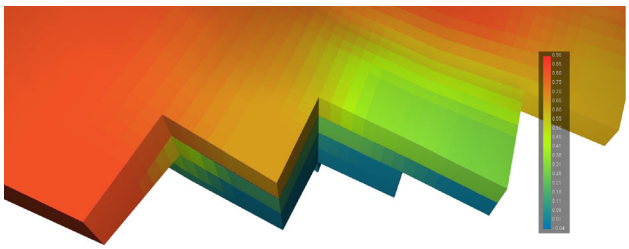
at each rendered mesh vertex. At visualization time, we can specify a 3D texture, here named the *property texture*, whenever a new property or new property time step is requested. This 3D texture has $ni \times nj \times nk$ voxels, where each voxel of index i, j, k holds the property value associated with the corresponding cell in the original model. The property is defined only for active cells.

Each fragment generated by the rasterization of an external face must be painted according to the property value of its associated cell in the original mesh. As done with the wireframe, we can use the topological coordinates i, j, k associated with each vertex to index the property texture. (Texture filters must be disabled.) When rendering the model external surface, the topological coordinates are adjusted to avoid accessing absent texels or texels associated with inactive cells. The topological coordinate that is constant throughout the face is offset to the center of the cell: 0.5 is added or subtracted according to the face orientation. This offsetting is implemented in a geometry shader. In the example in Fig. 11, the i coordinate has to be adjusted to 2.5.

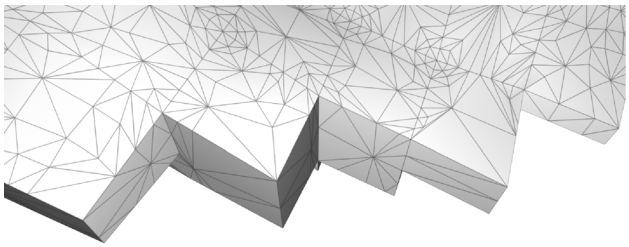
Figure 13 presents the visual results obtained by mapping the original property over the simplified meshes: Fig. 13a shows the property mapped over the original model; Fig. 13b shows the simplified meshes; Fig. 13c shows the property mapped over the simplified meshes. As can be noted, the first and third images are nearly identical.

5.3 Normal mapping

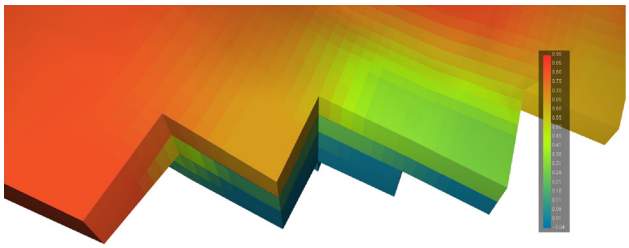
Surface lighting is also important to reveal shape. If we used the polygon normals of simplified meshes, the illumination



(a) Property mapped over original mesh



(b) Simplified meshes used on (c)



(c) Property mapped over the simplified meshes

Fig. 13 Property texture mapping

would become coarse as the meshes. We propose solving this by mapping original surface normals of top and bottom faces onto the simplified meshes. This is done in the same way as regular properties, allowing us to implement pixel-based illumination. In preprocessing time, original mesh normals on top and bottom faces of each cell are computed. They are stored with 2 bytes each, using the unit normal quantization method by Baptista [2]. In visualization time, normals can be mapped by building a 3D texture with four components for each cell, x, y for the top face normal and x, y for the bottom face normal. The z coordinate is derived at the fragment shader exploiting the fact of it being a unit normal and pointing up (top face) or down (bottom face). We only employ normal mapping on top and bottom faces; for side faces, we adopt the normals extracted from the simplified meshes, since these meshes are mostly plane.

5.4 Texture packing

If properties and normals were stored in complete 3D textures, scalability would suffer. If each layer is viewed separately, it is simple to use a 2D texture, which presents no problems in terms of graphics memory. For the visualization of the whole model, properties and normals are mapped only on the extracted external faces. A very large number of voxels are never accessed. We propose employing the Perfect Spatial Hashing technique by Lefebvre and Hoppe [9]. It packs sparse data into a compact table, in our case, a cubic 3D texture containing only the voxels that can be accessed in the rendering stage—those that are intersected by any extracted external face in the i, j, k coordinate space. A second, smaller 3D texture contains an offset table. Both the offset table and the location of original data in the compact 3D texture are computed in order to avoid any collisions, hence a perfect hash. These offsets and locations are also computed to maintain good spatial coherence, increasing the chance that two samples that are close in the original space are also close in the compact table. Given an i, j, k coordinate and the two 3D texture dimensions, it is very efficient to index the offset table and then the associated voxel.

The inclusion of this memory optimization improved graphics memory usage by an order of magnitude while preserving good performance. It allows some very big models to be viewed in the first place.

5.5 Decoupled visualization

The techniques described in this section allow the reuse of the generated hierarchical structure for a reservoir mesh on many different simulations of the same model. This is possible because the model partitioning, simplification, storage and the used perfect hash only depend on the reservoir topology and geometry. An actual simulation property and/or property time step can be loaded and stored separately, even at visualization time. This is particularly useful in the day by day engineering work, where many simulations are performed using the same base model grid.

6 View-dependent visualization

6.1 Data layout

Our viewer application supports the iso-contouring of reservoir cells, either viewing the whole model or each layer separately. As other out-of-core viewers, all extracted triangle meshes are compressed and concatenated into one big binary file, with pointers to mesh offsets and sizes stored in a separate file. The binary file is mapped to memory when

the visualization engine starts. Our implementation uses the LZO [12] library for lossless compression.

The binary file is first divided by layer, and then by triangular region. The meshes for each individual layer come first, and then the meshes of the whole model. For each mesh vertex, we store a position and an interpolated topological coordinate, summing 24 bytes for 6 floating point values. The top, bottom and side faces are stored separately. We compute normal cones [15] for the top and bottom meshes, allowing us to perform quick back-face culling.

6.2 Projected error measurement

Mesh selection in visualization time is made by projecting their simplification errors onto the screen. We use the method proposed by Lindstrom [10] to convert quadric errors into units of distance. A consistent upper bound for the projected error is the apparent size of a sphere, centered at the mesh bounding volume point closest to the viewer, with radius set to the square root of the quadric error.

Similar to the original Adaptive TetraPuzzles work [5], we store bounding volumes (BVs) and model space errors for each triangular region in a diamond-per-diamond basis. They are made the same for both triangular regions in each diamond: BVs enclosing the meshes of both regions and the model space errors to be the maximum between the two associated meshes. BVs and errors also monotonically decrease when descending the hierarchy. This enables a simple stateless top-down traversal of the binary trees [5] in visualization time. Given a projected error tolerance, the decision to split a node or not will always force the 2D space subdivision to be conformal, guaranteeing correct matching of mesh patches.

6.3 Real-time rendering

Our renderer is initialized by memory-mapping the binary file and spawning a prediction/load thread. At every frame, both the rendering thread and the prediction/load thread receive the current frame parameters (view frustum and screen space error tolerance). The prediction/load thread tries to predict the view frustums of the next frames and hint the memory map to load meshes that are not yet loaded. Meshes loaded from disk are transferred to the GPU in this separate thread, marking them as fully loaded when the process is complete. A LRU-based memory budget is also implemented in this thread to limit memory usage.

Part of the rendering effort is geometry processing, so we assume the frame rate is in part inversely proportional to primitive count. The primitive count itself is inversely proportional to the projected error tolerance. Given a target frame rate and based on previous rendering times, we indirectly adjust the primitive count by adjusting the projected error tolerance. This allows our viewer application to maintain the target frame rate, effectively applying geometric detail where it is most needed, given the available hardware.

7 Experimental results

We have tested our system using two computers. The first computer handled the out-of-core partitioning, simplification and packing of our multiresolution structure, equipped with two Intel Xeon Silver processors with 48 2.1 GHz threads and 256 GB of RAM. The second computer was used to view the models in our out-of-core multiresolution renderer. It is equipped with a 3.0 GHz Intel i7 processor with 16 GB of RAM and a NVIDIA Geforce GTX 660 GPU with 2 GB of graphics memory. The hierarchy bottom-up simplification was performed by 16 processes splitting the diamond simplification work on each hierarchy level. The extraction of external faces for iso-contour and simplification of side faces was done by 24 processes in parallel. All solution components were implemented in C++ and OpenGL.

Our solution was tested on three reservoir models with refinements ranging from 100 million to 1.2 billion cells. Table 1 shows the model sizes and preprocessing times. The third column shows the time spent partitioning the model and saving the leaf meshes. The time spent performing the hierarchy bottom-up simplification, external faces extraction and side faces simplification is shown in the fourth column. The fifth column shows the time spent packing the final binary file and computing the perfect hash for 3D textures.

Table 2 shows the multiresolution file sizes for all the models. The use of perfect hashing for our 3D textures was also evaluated: the third and fourth columns present the total sizes for full and packed 3D textures. The full size comprises $ni \times nj \times nk$ voxels, each using 2 bytes for the property scalar and 4 bytes for the top and bottom face normals. The packed size sums the storage for the two textures: (1) perfect hash offsets; (2) compact voxel table, also with 6 bytes per voxel. The percentages show that employing perfect hashing

Table 1 Model sizes and preprocessing times for partitioning, simplifying and packing the model

Model name	Cell count (M)	Part. time (h)	Simpl. time (h)	Packing time (h)	Total time (h)
A100	96	0.7	2.8	0.5	4.0
A217	217	1.6	3.9	1.1	6.6
B100	165	1.0	4.1	0.7	5.8
B240	378	2.3	7.5	1.8	11.6
C1200	1265	12.0	17.4	8.1	37.5

Table 2 Multiresolution structure, full and packed 3D texture sizes

Model name	Multiresolution structure file size (GB)	Full 3D texture sizes (MB)	Packed 3D texture sizes (MB)
A100	17.8	1094	96 (8.8%)
A217	38.9	2463	161 (6.6%)
B100	15.1	1816	136 (7.5%)
B240	39.6	4163	266 (6.4%)
C1200	181.0	13,938	491 (3.5%)

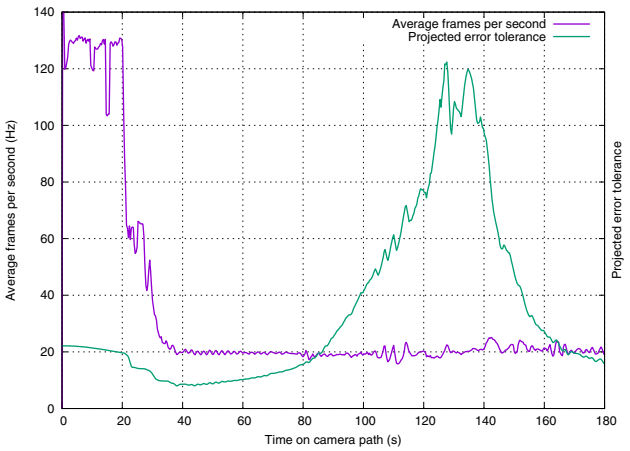


Fig. 14 Guarantee of a 20 FPS minimum frame rate during a navigation over the C1200 model, which originally contains 1.2 billion cells

has reduced the graphics memory usage by more than one order of magnitude, guaranteeing system scalability.

The real-time system performance was evaluated with our biggest model, setting the target frame rate as 20 FPS. A camera path with model manipulation and close navigation to inspect the model was used on the test. We measured the average frame rate and the projected error tolerances used during the path, shown in Fig. 14. It shows the system allowing the visualization of a 1.2 billion cell model interactively. The tolerances set for projected error during the path have successfully guaranteed at least 20 FPS during the whole path.

8 Conclusion and future work

We have presented a new out-of-core multiresolution system for rendering very large black oil reservoir models, a demand of the oil industry for numerical simulations with highly discretized domains. Experimental results have shown the effectiveness and scalability of the proposed solution. We highlight the following features of our proposal:

- A new simplification algorithm tailored specifically for reservoir meshes, preserving good hexahedral shapes and boundary face approximations;
- A novel out-of-core multiresolution structure for reservoir grids, decoupled from simulation properties, allowing its reuse in different simulations using the same geometry model;
- Texture construction and mapping techniques that allow the rendering of original model properties, wireframe and surface normals on top of the simplified meshes.

Our proposal has focused on the iso-contour visualization of properties mapped on the mesh external faces. However, we believe that other conventional visualization techniques can be easily adapted since we preserve topological columns and hexahedral cells. In future work, we plan to investigate the application of arbitrary cutviews and selection of region of interest on the simplified meshes. We also plan to explore the use of GPU to efficiently implement such algorithms. We also plan to investigate performance improvements of the partitioning and simplification stages.

This work was carried out consulting reservoir specialists; however, no formal evaluation was conducted. A qualitative end-user evaluation is left as future work, focused on achieved rendering quality and overall applicability in daily industry workflows.

Acknowledgements Tecgraf/PUC-Rio is a research institute mainly funded by Petrobras. This research was initiated during the Doctoral Program of the first author, financially supported by CNPq (Brazilian National Research and Development council).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abraham, F., Celes, W.: Distributed visualization of complex black oil reservoir models. In: Eurographics Symposium on Parallel Graphics and Visualization (EGPGV09), pp. 87–94. Eurographics Association (2009)
2. Baptista, R.: Higher Accuracy Quantized Normals. https://www.gamedev.net/articles/programming/math-and-physics/higher-accuracy-quantized-normals-r1252. Accessed in Feb. 2019
3. Blacker, T.D., Stephenson, M.B.: Paving: a new approach to automated quadrilateral mesh generation. Int. J. Numer. Methods Eng. **32**(4), 811–847 (1991)
4. Celes, W., Abraham, F.: Fast and versatile texture-based wireframe rendering. Vis. Comput. **27**, 939–948 (2011)
5. Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., Scopigno, R.: Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. In: SIGGRAPH'04: International Conference on Com-

puter Graphics and Interactive Techniques, pp. 796–803. ACM Press (2004)
6. Dake, L.P.: Fundamentals of Reservoir Engineering. Elsevier, Amsterdam (1978)
7. Daniels, J., Silva, C.T., Shepherd, J., Cohen, E.: Quadrilateral mesh simplification. ACM Trans. Graphics **27**, 148:1–148:9 (2008)
8. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: SIGGRAPH'97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, pp. 209–216. ACM Press, New York (1997)
9. Lefebvre, S., Hoppe, H.: Perfect Spatial Hashing. ACM Trans. Graphics **25**(3), 579–588 (2006)
10. Lindstrom, P.: Out-of-core construction and visualization of multiresolution surfaces. In: Symposium on Interactive 3D Graphics, pp. 93–102. ACM Press, New York (2003)
11. Liu, H., Chen, Z.: A scalable thermal reservoir simulator for giant models on parallel computers. CoRR (2018). arXiv:1812.03952
12. Oberhumer, M.F.X.J.: LZO Data Compression Library. http://www.oberhumer.com/opensource/lzo. Accessed in Feb. 2019
13. Saudi Aramco Completes First Giga-Cell Reservoir Simulation Run. http://www.rigzone.com/news/article.asp?a_id=70015. Accessed in Feb. 2019
14. Shepherd, J.F., Dewey, M.W., Woodbury, A.C., Benzley, S.E., Staten, M.L., Owen, S.J.: Adaptive mesh coarsening for quadrilateral and hexahedral meshes. Finite Elem. Anal. Design **46**(1–2), 17–32 (2009)
15. Shirman, L., Abi-Ezzi, S.: The cone of normals technique for fast processing of curved patches. In: Eurographics, pp. 261–272. Eurographics Association (1993)
16. Sousa, M.C., Vital Brazil, E., Sharlin, E.: Scalable and inter active visual computing in geosciences and reservoir engineering. Geol. Soc. **406**(1), 447–466 (2015)
17. Staten, M., Canann, S.A.: Post refinement element shape improvement for quadrilateral meshes. ASME Trends Unstruct. Mesh Gen. **220**, 9–16 (1997)
18. Staten, M.L., Benzley, S., Scott, M.: A methodology for quadrilateral finite element mesh coarsening. Eng. Comput. **24**, 241–251 (2008)
19. Tarini, M., Pietroni, N., Cignoni, P., Panozzo, D., Puppo, E.: Practical quad mesh simplification. Comput. Graph. Forum. (Special Issue of Eurographics 2010 Conference) **29**(2), 407–418 (2010)
20. Weiss, K., De Floriani, L.: Diamond hierarchies of arbitrary dimension. Comput. Graph. Forum **28**(5), 1289–1300 (2009)
21. Wu, J., Kobbelt, L.: Fast mesh decimation by multiple-choice techniques. In: Vision, Modeling and Visualization, pp. 241–248. IOS Press, Amsterdam (2002)

Frederico Abraham is a researcher and developer at Tecgraf/PUC-Rio Institute, where he works on scientific visualization projects. He received his B.S., M.S. and D.S. degrees in computer science and computer graphics in PUC-Rio, the Pontifical Catholic University of Rio de Janeiro, Brazil. His current research interests include real-time rendering, multiresolution rendering, scientific visualization and distributed visualization.

Waldemar Celes is an associate professor of computer science at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro, Brazil, and an associate researcher at Tecgraf/PUC-Rio Institute, where supervises projects on scientific visualization and numerical simulation in cooperation with the industry. He is an co-author of the Lua programming language.