Parallel extraction and simplification of large isosurfaces using an extended tandem algorithm

Guilhem Dupuy, Bruno Jobard, Sebastien Guillon, Noomane Keskes, Dimitri Komatitsch

Parallel Extraction and Simplification of Large Iso-surfaces using an Extended Tandem Algorithm

Guilhem Dupuy [1] Bruno Jobard [1,3] Sebastien Guillon [4]
Noomane Keskes [4] Dimitri Komatitsch [2,3]

Abstract

In order to deal with the heavy trend in size increase of volumetric datasets, in the past few years research in isosurface extraction has focused on related aspects such as surface simplification and load-balanced parallel algorithms.

We present a parallel, bloc-wise extension of the tandem algorithm [1], which simplifies on the fly an isosurface being extracted. Our approach minimizes overall memory consumption using an adequate block splitting and merging strategy along with the introduction of a component dumping mechanism that drastically reduces the amount of memory needed for particular datasets such as those encountered in geophysics. As soon as detected, surface components are migrated to the disk along with a meta-data index (oriented bounding box, volume, etc) that permits further improved exploration scenarios (small components removal or particularly oriented components selection for instance).

For ease of implementation, we carefully describe a master and worker algorithm architecture that clearly separates the four required basic tasks. We show several results of our parallel algorithm applied on a geophysical dataset of size $7000 \times 1600 \times 2000$.

Key words: Geometric computation, Distributed design and Geo-scientific applications

Email addresses: guilhem.dupuy@univ-pau.fr (Guilhem Dupuy), bruno.jobard@univ-pau.fr (Bruno Jobard), sebastien.guillon@total.com (Sebastien Guillon), noomane.keskes@total.com (Noomane Keskes), dimitri.komatitsch@univ-pau.fr (Dimitri Komatitsch).

[1] LIUPPA: Computer Science Laboratory of the University of Pau, France
[2] MIGP: Geophysics Laboratory of the University of Pau, France
[3] INRIA-Magique3D
[4] Total

1 Introduction

Surface reconstruction for shape modeling is widely used in a large variety of fields (medicine, geophysics, ...). The marching cubes algorithm, introduced by Lorensen and al. [2], is the most classical algorithm used for isosurface extraction. Due to the increasing size of processed datasets and extracted surfaces, many improvements of the marching cubes have been proposed. They concern for instance ambiguities treatment [3], reduction of the number of traversed cells [4, 5], load balancing in parallel approaches [6, 7] and reduction of the number of generated triangles [8].

In order to cope with the increasing size of datasets, isosurfaces might need to be simplified to reduce the number of generated triangles both for memory storage or more importantly for visualization purposes. In a brute-force approach one would extract the full mesh and simplify it in a second pass [9] [10] [11]. The problem with this method lies in the generation of a first very large mesh that may not fit in the main memory before further simplification. In [1], Attali et al. reduce memory requirements by introducing a tandem algorithm that combines isosurface extraction and simplification stages in one pass. Their method drastically reduces the amount of vertices and triangles stored in memory during extraction, allowing larger datasets to be processed.

Similar to how Attali et al. addressed the memory problem raised by larger datasets, we introduce a parallel, bloc-wise extended version of the tandem algorithm to accelerate the computation of simplified isosurfaces. The dataset is split in blocks and sent to compute nodes. In each node a local isosurface is extracted and semi-simplified based on a slightly modified tandem algorithm. Then nodes can receive an adjacent semi-simplified isosurface that will be merged with the local one. This merge operation ends with a simplification stage with relaxed edge constraints at their common interface that remove seams between them. The algorithm finishes when all local semi-isosurfaces have been merged and simplified. Our splitting/merging strategy forces adjacent nodes to be processed together, maintaining memory consumption as low as possible during the overall isosurface extraction.

We also introduce an early components dumping mechanism that frees the memory as soon as independent surface components have been extracted, simplified and stored to disk along with meta-data for later high-level exploration. This strategy has proven to be very useful for particular datasets such as in geophysics for which the extracted features are numerous but small compared to the global size of the volume. The meta-data stored along these disconnected components can be used for filtering purposes during their visualization, for instance discarding those with too small volumes or keeping particularly aligned ones. This dumping mechanism can also be beneficial for

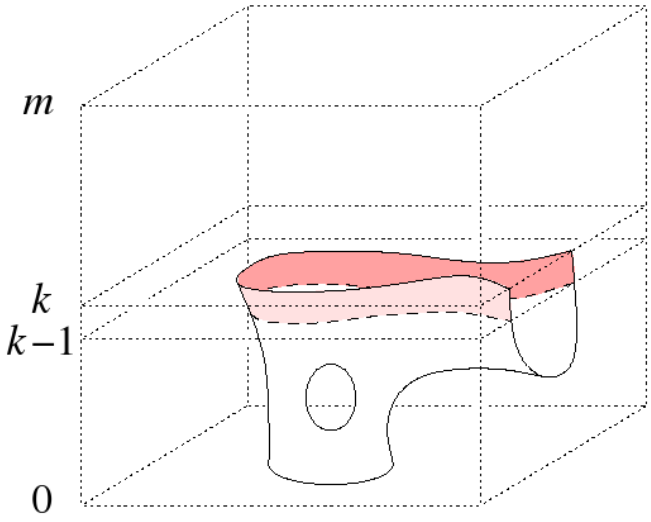


Fig. 1. The k-th layer (in red) is the set of vertices, edges and triangles extracted with the marching cubes algorithm between cross-sections k-1 and k. Figure courtesy of Dominique Attali.

noisy datasets or when a pertinent isovalue is not yet well determined and leads to many small disconnected components.

In the first part of this article we describe the tandem algorithm. In the second part we propose a parallel extension of this algorithm with dumping of completed objects. The third part presents some computational experiments.

2 The "Tandem Algorithm"

The main idea of the algorithm of Attali et al. [1] is to alternate the extraction of a layer (see figure 1) and the simplification of the current overall extracted surface in order to reduce the amount of occupied memory. Indeed, a global simplification after a complete extraction would require to store all vertices and triangles, while a simplification stage during extraction reduces the number of vertices and triangles at each step.

The tandem algorithm is then a simple loop that iterates over the cross-sections to implement two operations:

- an "extraction" operation that adds a layer (vertices, edges and triangles obtained with the marching cubes algorithm between two subsequent cross-sections) to the current triangulation.
- a "simplify" operation that simplifies the current triangulation. The last layer added is not simplified so that the next layer can be appended.

The simplification stage consists in applying the classical edge collapse algorithm [12]. Each edge collapse operation has a cost that measures the numerical error it would introduce in the triangulation. Edge candidates for edge collapse are kept in a priority queue $\mathcal{Q}$ ordered by their cost. To evaluate this cost Attali et al. [1] revisited the quadratic metric of Garland and Heckbert [12] and proposed a quadratic error which is a weighted sum of a shape measure criterion and a mesh isotropy criterion.

The shape measure is similar to that used in the original edge contraction algorithm. It measures the deviation introduced by collapse operation between the new vertices and the original surface. The shape measure of a point x is defined by

$$h_c(x) = \frac{1}{W_c} \sum_{t \in U_c} w_t d^2(x, P_t) = \frac{1}{W_c} x^T \mathbf{H}_c x$$

where U_c is the patch defined by all the neighboring triangles of point c. P_t is the plane spanned by the triangle t, w_t its area and $W_c = \sum_{t \in U_c} w_t$. $\mathbf{H}_c$ is a positive definite matrix. Edge contraction $ab \mapsto c$ leads to $\mathbf{H}_c = \mathbf{H}_a + \mathbf{H}_b$ and $W_c = W_a + W_b$.

The mesh isotropy criterion is introduced in order to prevent the creation of long and skinny triangles. Considering S_{ab} the set of triangles containing the vertices a, b or both in the current triangulation, the mesh isotropy criterion for point c is defined as the squared distance of the point to the patch:

$$g_c(x) = \sum_{t \in S_{ab}} w_t(||x - \hat{t}||^2 + avg(t)) = x^T \mathbf{G}_c x$$

where $\hat{t}$ is the barycenter of the triangle t and $avg(t) = \frac{1}{12}(||p||^2 + ||q||^2 + ||r||^2)$ with p, q, r the vectors from $\hat{t}$ to the vertices of t. The term g_c is normalized by $W = 3 \times area(S_{ab}) W_c^{1/2} / E_0$ in order to balance its influence with h_c in the global cost defined by :

$$\varepsilon_\alpha(c) = \sqrt{c^T [(1-\alpha)\frac{\mathbf{H}_c}{W_c} + \alpha \frac{\mathbf{G}_c}{W}] c}$$

α is called the isotropy parameter, and represents a compromise between the shape measure criterion and the anisotropy measure criterion. In practice α is set to 0.4 for a good compromise between the two criteria.
For more details about those mathematical formulations, please refers to the original article [1].

During edge contraction, the resulting vertex position c is obtained by minimizing the local error function ε_α and the final error value $\varepsilon_\alpha(c)$ is used to order the priority queue $\mathcal{Q}$. In any case, a candidate cannot be accepted if its shape measure, $\varepsilon_0(c)$, exceeds a positive constant error threshold E_0. The *simplify* function then consists in emptying the queue $\mathcal{Q}$ by applying consecutive

edge collapses.

Right after the extraction stage, the simplification stage has to cope with the heavy edge constraints on the last extracted layer. To prevent the creation of artifacts that these blocked edges would introduce, an innovative way of scheduling the edge collapse was proposed, called time lag. The main idea was to delay edge collapses near the advancing front. The time lag is based on the rank of a vertex, equal to its coordinate along the extraction direction (for example z if the cross-sections are taken along the z-axis). The rank of front is the maximum rank that has been extracted. Considering, $height(u) = rank(u)$ and $rad(u) = 1$ for new vertices introduced by extraction, the contraction of an edge $ab \mapsto c$ leads to :

$$\begin{aligned} height(c) &= (height(a) + height(b))/2 \\ rad(c) &= (||a - b|| + rad(a) + rad(b))/2 \\ reach(c) &= height(c) + rad(c) \end{aligned}$$

The contraction of an edge is prevented as long as its reach value is greater or equal to the rank of the advancing front. As detailed in [1], if a and b belong to the last extracted layer, height(c) = rank(front) and since rad(c) > 0, the contraction of ab would be prevented. Similarly, if a vertex of ab lies in the front plane, reach(c) would be greater than the rank of the advancing front.

The blocked edges are kept in a priority queue $\mathcal{W}$ ordered by reach value. The function *delay* adds all edges of the last layer k in $\mathcal{W}$. An edge is moved from $\mathcal{W}$ to $\mathcal{Q}$ if its reach value is lesser than the rank value of the advancing front. The function *activate*, described in algorithm 1, moves edges from $\mathcal{W}$ to $\mathcal{Q}$. Figure 2 illustrates the effect of the time lag near the advancing front.

Procedure *activate*(k : **integer**)
  While ($reach(top(\mathcal{W})) <$ k) **do**
    add $top(\mathcal{W})$ in $\mathcal{Q}$;
    $pop(\mathcal{W})$;
  done
End

Algorithm 1: *At the k-th layer, the "activate" function fills up the edge collapse candidates queue $\mathcal{Q}$ with previously delayed edges of $\mathcal{W}$.*

.

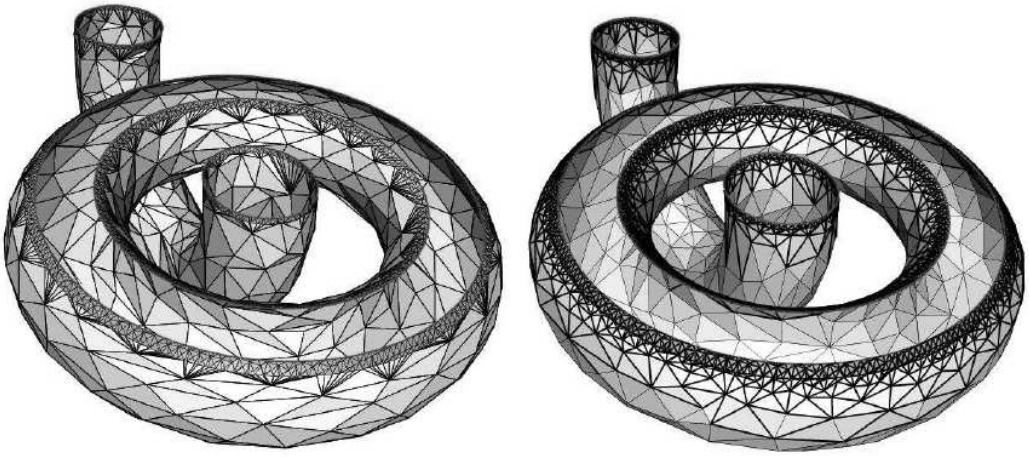


Fig. 2. Two partial triangulations constructed without (left) and with (right) the time lag: the length of the edges gets progressively longer as they get further to the front layer. Figure courtesy of Dominique Attali

The tandem algorithm can then be written as in algorithm 2. It takes E_0, the maximum shape error allowed, as a parameter.

```
Procedure tandem( E0 : float)
    For k from 1 to number of cross-sections - 1 do
        extract(k);
        delay(k);
        activate(k);
        simplify(E0);
    end For
    activate(∞);
    simplify(E0);
End
```

Algorithm 2: *The tandem algorithm*

3 Extended tandem algorithm

The tandem algorithm was designed to work on large datasets but experimental results (a test that we performed on a noisy dataset of size 1626×7028×2000) showed us the limited scalability of this algorithm. On this cube, each extraction on the advancing fronts generates 1 250 000 triangles (figure 3). In this case, updates of the queues and simplification steps are too memory consuming. We therefore propose an extension of the tandem algorithm to increase its scalability. This extension consists in dumping parts of the extracted surface and dispatching the extraction process on subsets of the dataset.

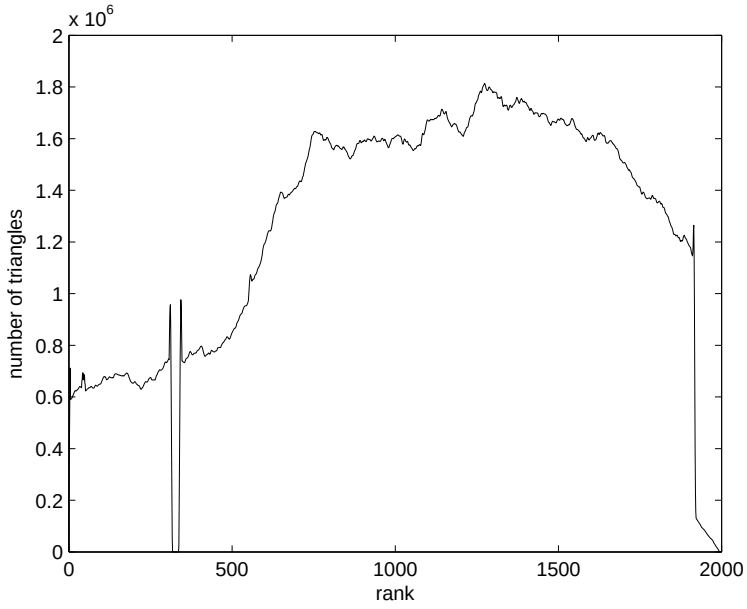


Fig. 3. Number of triangles generated for each layer of the (1626×7028×2000) dataset. The region with no triangles extracted is due to a hole in the dataset.

3.1 Components dumping

Classical out-of-core approaches migrate mesh to disk during extraction. The generated files are then soups of triangles ordered by direction extraction. But isosurfaces extraction, especially in geophysics, generate many disconnected components. Figure 4 illustrates an usual repartition of geological sets: finished components are shown in light gray and components that are still growing or that are not completely simplified are shown in dark gray. This distribution of disconnected components allows us to dump completed surfaces as soon as their simplification is finished.

The dumping process requires to detect the termination of a component ex-

Fig. 4. Distribution of extracted components. Finished components are shown in light grey and growing ones in dark grey. The black arrow indicates the direction used for extraction.

traction, but since we use the time lag technique, a component may be entirely extracted but its simplification may have been prevented. We therefore need to detect the end of its extraction as well as the end of its simplification.

To formalize the finalization of a component, we define an active component γ as the set of vertices defining its shape. We consider Γ the set of all the active components, $\Gamma = \{\gamma_i\}$ and define V_w the set of all the vertices defining edges in the queue $\mathcal{W}$ (all the edges prevented by the time lag technique). A component is then finalized if it has no more edges to collapse (γ has no more edges in $\mathcal{W}$), which can be written as :

$$\gamma \text{ is finalized} \Leftrightarrow \gamma \cap V_w = \emptyset$$

We define the function *save*(γ) that migrates a component to disk and the function *clear*(γ) that deletes γ in the current triangulation. The *Dumping* function can then be written as in algorithm 3.

```
Procedure dumping()
    For Each γ in Γ do
        If (γ ∩ V_w = ∅) then
            save(γ);
            clear(γ);
        end If
    end For
End
```

Algorithm 3: *The "dumping" function*

The *dumping* function is then introduced in the tandem algorithm (algorithm 4).

```
Procedure tandem( E_0 : float)
    For k from 1 to number of cross-sections - 1 do
        extract(k);
        delay(k);
        activate(k);
        simplify(E_0);
        dumping();
    end For
    activate(∞);
    simplify(E_0);
    dumping();
End
```

Algorithm 4: *The new tandem algorithm with dumping*

An intrinsic property of our component-based dumping is that our generated file is ordered by component. We can then easily access a subset of components by reading a subset of the generated file. To optimize access to components in

this file (called the raw data file) we generate an index file (called the index component file). To enhance the exploration of extracted surface, meta-data (such as oriented bounding box, volume, number of vertices and facets, ...) are computed for each component and stored in the index file (figure 5). One exploration scenario could be based on volume filtering such as in [13] in which Pivot et al. propose a workflow for complex volume seismic interpretation. They suggest to extract isosurfaces on seismic attributes and to delete automatically inconsistent small "bubbles" (disconnected components with very small volumes with respect to other components). Another useful way of separating components is an analysis based on the geological depositional direction (azimuth direction).

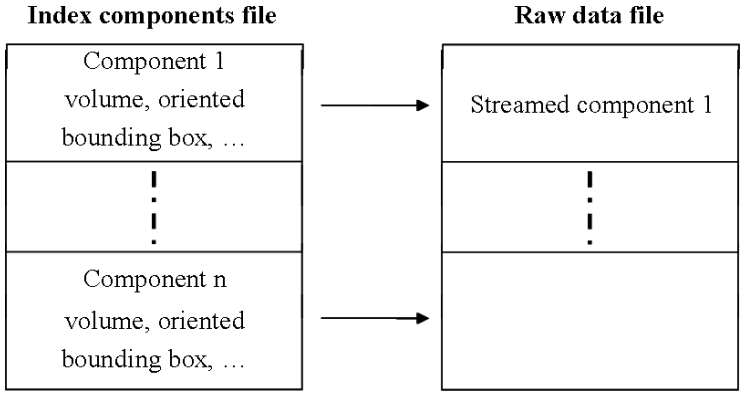


Fig. 5. File organization. The index file (left) refers to the raw data file (right).

3.2 Parallel algorithm

A good feature of the marching cube is its spatial independence. Each grid cell can be processed independently to the others and in any traversal order. We therefore propose to split datasets in subsets, to extract simplified surfaces in these subsets using the extended tandem algorithm and to merge them in a final surface.

Our parallel algorithm considers the dataset as a layout of subsets on which a slightly modified tandem algorithm is first applied. In fact, during the continuous simplification of the extracted triangulations, edge collapse operations are not applied near boundaries in order to enable a later merge with adjacent blocks of iso-surfaces. Blocks of semi-simplified iso-surfaces are then aggregated as they become available and aggregates get simplified again to remove seams in the merged triangulation.

Dealing with semi-simplified iso-surface blocks must imply a high priority in merging them as soon as possible to keep the overall number of triangle as low as possible. The splitting strategy and the subset traversal order must be designed with this requirement in mind.

A classical dataset subdivision as a regular grid of blocks seems natural for this purpose, but it becomes tricky to choose along time the next good block to process and with which one it is better to merge. To address this aspect we have chosen to complement this grid with an implicit hierarchical organization scheme that allows a fast selection of adjacent blocks without traversing the whole grid of blocks. Process and merge orders are directed thank to this hierarchical layout.

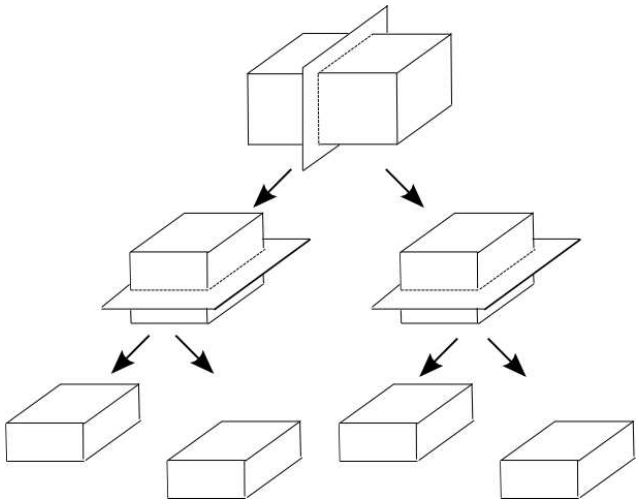


Fig. 6. Binary partitioning. The dataset is recursively split perpendicularly to its longest direction until it goes down a given size threshold.

Splitting phase We propose a hierarchical scheme for surface extraction. As in [14], the dataset is split recursively in two parts perpendicularly to its longest direction as long as the sub-block size is greater than a given threshold (figure 6). These successive subdivisions of the dataset implies a binary tree logical organization (figure 7). Isosurfaces extraction are performed on leaves of the generated tree and recursively merged to reconstruct the global surface.

Extraction in leaf nodes is performed using the tandem algorithm combined with our new dumping approach. The size of the leaves could then be relatively

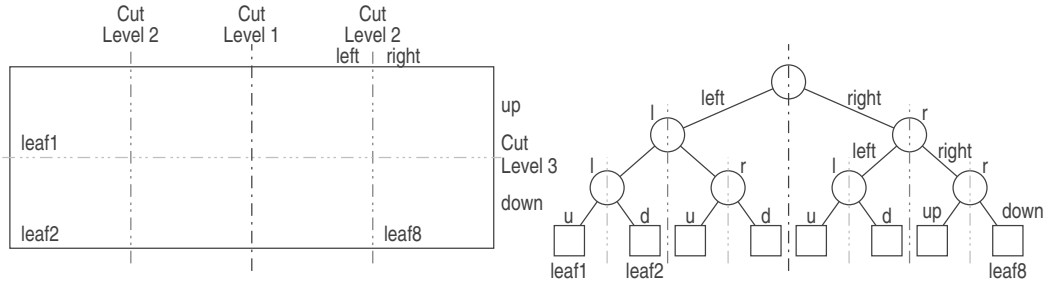


Fig. 7. Binary partitioning of a dataset (left) and the resulting logical organization as a binary tree (right). Each node of the tree is tagged by a type indicating its position regarding to the cutting plane.

large since the tandem algorithm has been designed for this purpose. However our parallel algorithm is quite independent of this size. In our experiments, overall extraction timings do not significantly vary with 128^3, 256^3 or 512^3 leaf block sizes.

In order to avoid refinement dependencies between adjacent blocks and prevent artifacts due to a bias in shape during simplification, we extend the time lag notion. As in the original time lag the radius associated with a point c (result of the collapse $ab \mapsto c$) is given by

$$rad(c) = (||a - b|| + rad(a) + rad(b))/2$$

with $rad(x) = 1$ if x is an original vertex of the isosurface. The contraction of edge ab is prevented as long as the sphere centered on the middle of ab and of radius $rad(c)$ is not totally included in the block. We show in figure 9 how the introduction of the time lag influences the refinement progression near the boundaries of the blocks.

Merging phase In our splitting strategy (binary partitioning), each node p is split in two children $p1$ and $p2$, denoted c_p (children of p).Reciprocally p is the father of $p1$ and $p2$ is denoted by f_{p1} and f_{p2} respectively; $p1$ is the brother of $p2$ by $b_{p2} = p1$ and reciprocally $b_{p1} = p2$.

Intuitively, a good strategy for block merging should be to recursively merge brothers, from the leaves up to reaching the root of the tree. Considering brothers ensures that they share a large common boundary and that the simplification of the merged triangulation will remove a significant number of triangles. In practice iso-surface components are not uniformly distributed in the sub-blocks and therefore extraction time of two brothers could be drastically different, and merging them implies to wait for the slower one. To solve this problem we allow the algorithm to only merge a block with its brother or with one of its nephews if they have a common boundary.

We thus need to define a restricted adjacency criterion determining if a sub-block of a brother node has a common boundary. We consider that a block p can be split along three different planes (according to its longest direction). We define the type of a node as the position of the node after the split of its father. If a node p is split along x $p1$ and $p2$ are denoted respectively left and right. Similarly, cut along y defines children as up and down, and cut along z as front and back.
Denoting $\overline{\cdot}$ the opposite operator, we have:

$$\overline{\text{left}} = \text{right and } \overline{\text{right}} = \text{left}$$
$$\overline{\text{up}} = \text{down and } \overline{\text{down}} = \text{up}$$
$$\overline{\text{front}} = \text{back and } \overline{\text{back}} = \text{front}$$

We define *tree* of root p (T_p) the set of nodes containing p and its recursive children. In this tree, *ancestry* of a node n ($A_{p,n}$) is defined as the set of nodes containing n and its recursive fathers up to p:

$$A_{p,n} = \begin{cases} n & if\ n = p \\ n \cup A_{p,f_n} & else \end{cases}$$

Considering a block split in $p1$ and $p2$, then a nephew node n belonging to $T_{c_{p2}}$ has a common boundary with $p1$ if

$$\forall\ n' \in A_{c_{p2},n},\ \text{type}(n') \neq \overline{\text{type}(p1)}$$

If n then fulfills this restricted adjacency requirement it can be merged with $p1$. This criterion can efficiently be implemented via an iterative tree traversal through the successive fathers while verifying the condition on their types. When two brothers have been merged, their father node is considered completely processed.

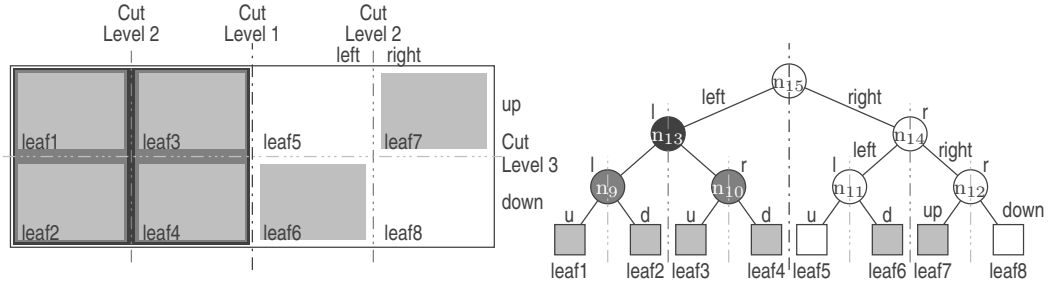


Fig. 8. Restricted adjacency requirement. Merging operations occur between brother nodes or between a node and its nephews that share a common boundary. Light gray leaves have already been processed. Darker gray nodes are considered as merged and completely processed.

To understand better these merging requirements, let's consider figure 8. Leaves 1 and 2 are processed and are brother nodes, they can then be merged; n_9 is then considered as completely processed. Same case for n_{10} with the leaves 3 and 4. It results that n_9 and n_{10} can also be merged and that n_{13} is then completely processed. n_{13} cannot be directly merged with n_{14} because n_{14} is not completely processed. However leaf6 can be merged with n_{13} because it fulfills the requirements on the types of its ancestry nodes ($A_{c_{\text{n}_{14}},\text{leaf6}} = \{\text{leaf6}; \text{n}_{11}\}$): $\text{type}(\text{leaf6}) = \text{down} \neq \overline{\text{type}(\text{n}_{13})} = \text{right}$ and $\text{type}(\text{n}_{11}) = \text{down} \neq \overline{\text{type}(\text{n}_{13})} = \text{right}$. At last, leaf7 cannot be merged with n_{13} because one of its ancestry nodes ($A_{c_{\text{n}_{14}},\text{leaf7}} = \{\text{leaf7}; \text{n}_{12}\}$) does not fulfills the requirement: $\text{type}(\text{n}_{12}) = \text{right}$ is not different to $\overline{\text{type}(\text{n}_{13})} = \text{right}$. This last case can be seen on figure 8, leaf7 does not have any common boundary with the dark node n_{13}.

Geometrically, merging two sub-blocks requires first to identify common points

extracted in each block lying on their shared boundary. Due to potential numerical inaccuracy in vertices extraction, we cannot identify common points based on a simple coordinates comparison. Identification is then done by using a property of the marching-cubes algorithm: as a cell edge can have only one or zero points generated by the algorithm, two points belonging to common cell edges are identical and then merged.
Once this merging step is finished, one has to simplify pasted area with the same error criterion as for the surface extraction E_0. As during the extraction step, the contraction of an edge ab is prevented as long as the sphere centered on the middle of ab and of radius $rad(c)$ is not totally included in merged blocks. We show in figure 9 the merge of surfaces and the effect of the time lag on the quality of resulting surfaces. Finally the last step of the merge consists in dumping finished components to the disk.

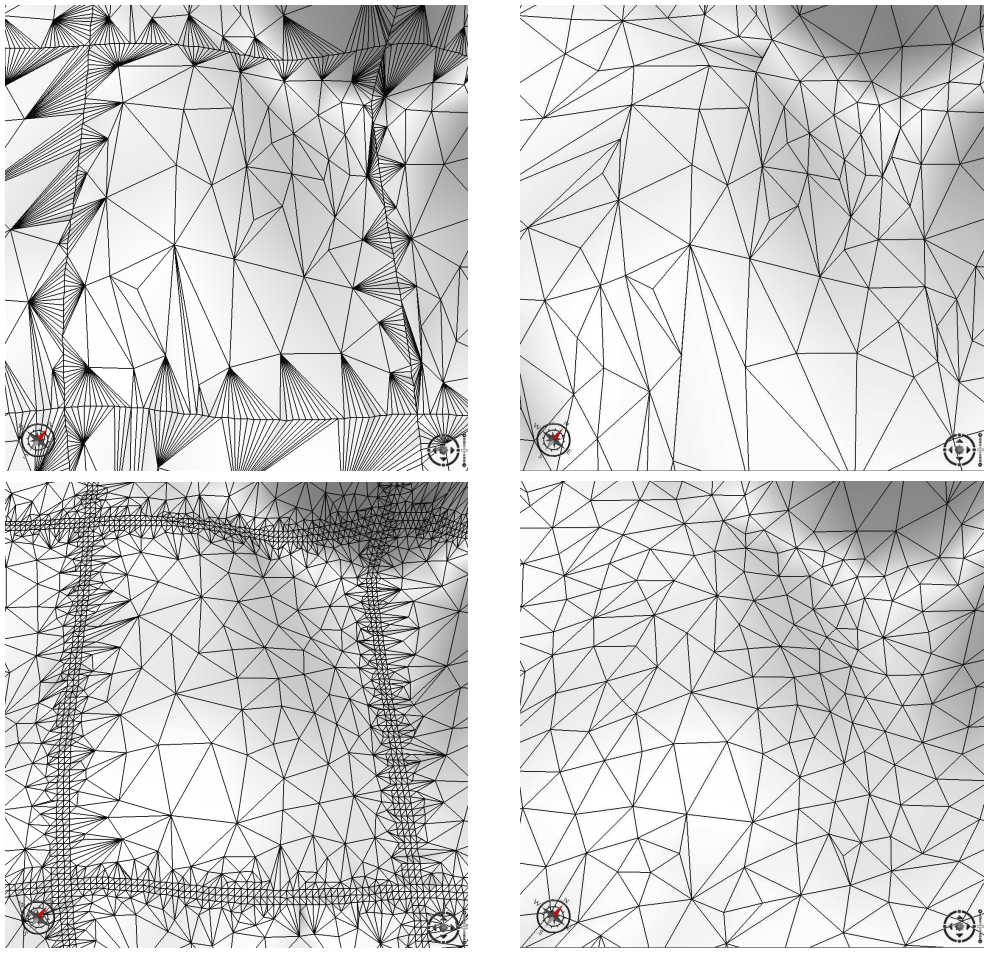


Fig. 9. (left) we illustrate the effect of the time lag on the refinement at the boundaries of the blocks before simplification. On the right, surfaces have been simplified. On the top row, only edges that have at least a vertex on a boundary have been kept. On the bottom row, edges have been excluded from simplification using the time lag method. This shows that applying the time lag method near the boundaries of blocks leads to triangulations of better quality.

In our application the target parallel machine is a *load-balanced cluster* managed by the Platform LSF-HPC software. A typical use of this parallel machine is to decompose a process in independent tasks and then let the task scheduler (here LSF-HPC) dispatch tasks to the distant grid. When a task is finished, the processor used for this task is freed. The task scheduler can reallocate it for the process if resources are sufficient for other running processes.

We propose a parallel algorithm adapted to this kind of architecture that is composed of workers and a worker manager. Workers are distant processes that have to extract, merge and dump surfaces or send them to other workers. The worker manager is a process, distant or not, that assigns tasks to workers. This approach and our formalism are adapted to our target parallel machine (number of allocated processor varies in time) but could also be used on other parallel machines. It is also worth to mention that our algorithm can also be run with no modification with only one worker. In this case, this worker will process and aggregate successively all the blocks of iso-surface in the grid.

Workers do not have any consciousness of the tasks they have already accomplished, the portions of surfaces they currently own or if any other worker is processing adjacent blocks, the worker manager does. Workers simply ask and obey to the worker manager that uses the hierarchical structure to quickly find what is the next best block to extract for a given worker or to what worker an other one should send its aggregate of iso-surfaces before its termination.

A worker is written as an infinite loop that requests tasks from the worker manager(algorithm 5). This loop ends when the worker manager decides so. Tasks are composed of a task type and a worker id (each worker has an unique id used to send messages). We distinguish four task types:

- *EXTRACT*: the worker has to extract a surface from a sub-block of the dataset. The extraction is performed with the extended tandem algorithm. Sub-block boundaries, excepted those that are common to dataset boundaries, would prevent the edge collapse in their neighborhood using time lag. The extracted surface is appended to its current surface.
- *SEND*: the worker received a worker id, referring to a target worker. The worker has to send its current mesh to the target worker that will merge it with its current one. After this send, the current surface of the worker is emptied.
- *MERGE*: the worker has to merge its current surface with a mesh sent by another worker. After the merge operation, the new current surface is simplified and finished components are dumped.
- *FINISHED*: the worker has to stop. The time life of a worker is not neces-

sarily equal to the global process duration.

```
Procedure worker( E0 : float)
    finished : boolean;
    finished ← false;
    While ( ¬finished) do
        task ← get_task();
        Switch
            task.type=EXTRACT:
            tandem(E0);
            task.type=SEND: send_surface(task.target);
            task.type=MERGE: merge_surface(get_surface(), E0);
            task.type=FINISHED:
            merge_surface(get_surface(), E0);
            finished ← true;
        end Switch
    done
End
```

Algorithm 5: *The worker algorithm. Each worker receives at most four distinct tasks from the worker manager.*

The worker manager is a loop that iterates as long as the global surface has not been totally extracted and recomposed. The worker manager has a global view of the process and can dynamically dispatch tasks to workers, which ask for tasks when they begin or when they finish their last assigned task. In order to define the worker manager mechanism (algorithm 6), we define the following three operations that operate efficiently on top of the hierarchical organization brought by the recursive splitting mechanism: *has_waiting_surface*, *has_neighbor_worker* and *has_unprocessed_block*.

Function *has_waiting_surface*: if a querying worker has some waiting surface sent by other workers it returns true, else false.

Function *has_neighbor_worker*: if it exists a worker that is dealing with a node that fulfills the restricted adjacency requirements with the aggregate of surfaces of the querying worker then it returns the id of this worker, else it returns false.

Function *has_unprocessed_block*: the worker manager finds here the best next block to extract for the querying worker. Among the unprocessed blocks, it will first look for one that fulfills the restricted adjacency requirements with the current aggregate of surfaces of the querying worker. If no such adjacent block exists it returns any unprocessed one. If no one exists it returns false.

Based on these three functions, the algorithm of the worker manager (algorithm 6) performs as follows. For each request received by workers, the manager asks first to merge any waiting surface. If no surface is waiting, it determines if the worker has a neighbor worker so it can send its aggregate of surfaces. If not, the manager asks the worker to extract a new unprocessed block. If no unprocessed block remains it ask the worker to finish.

```
Procedure worker_manager()
    While ( ¬finished) do
        id ← get_task_request_id();
        If (has_waiting_surface(id)) then
            send_task_to_worker(id, MERGE);
        else
            If (has_neighbor_worker(id) ) then
                send_task_to_worker(id, SEND, targetId);
            else
                If (has_unprocessed_block(blockId) ) then
                    send_task_to_worker(id, EXTRACT, blockId);
                else
                    send_task_to_worker(id, FINISHED);
                end If
            end If
        end If
    done
End
```

Algorithm 6: *The worker manager has a global view over every workers. Depending on their current state, it decides and sends them what is the next task they will accomplish.*

3.4 Example of algorithm trace

The purpose of this section is to show how the algorithm performs on an example. It will help us illustrate the concepts introduced in the previous sections.

Figure 10 shows a possible trace of our algorithm 6 on a dataset recursively split in eight blocks in the same way than in figure 7. In this trace we use the notation "Wi ? → task argument", which means "*worker number i asks the manager what to do now and the manager replies with a task to accomplish and its argument*". For clarity sake, several states of the dataset is displayed after groups of few accomplished tasks. This trace is obtained with three workers.

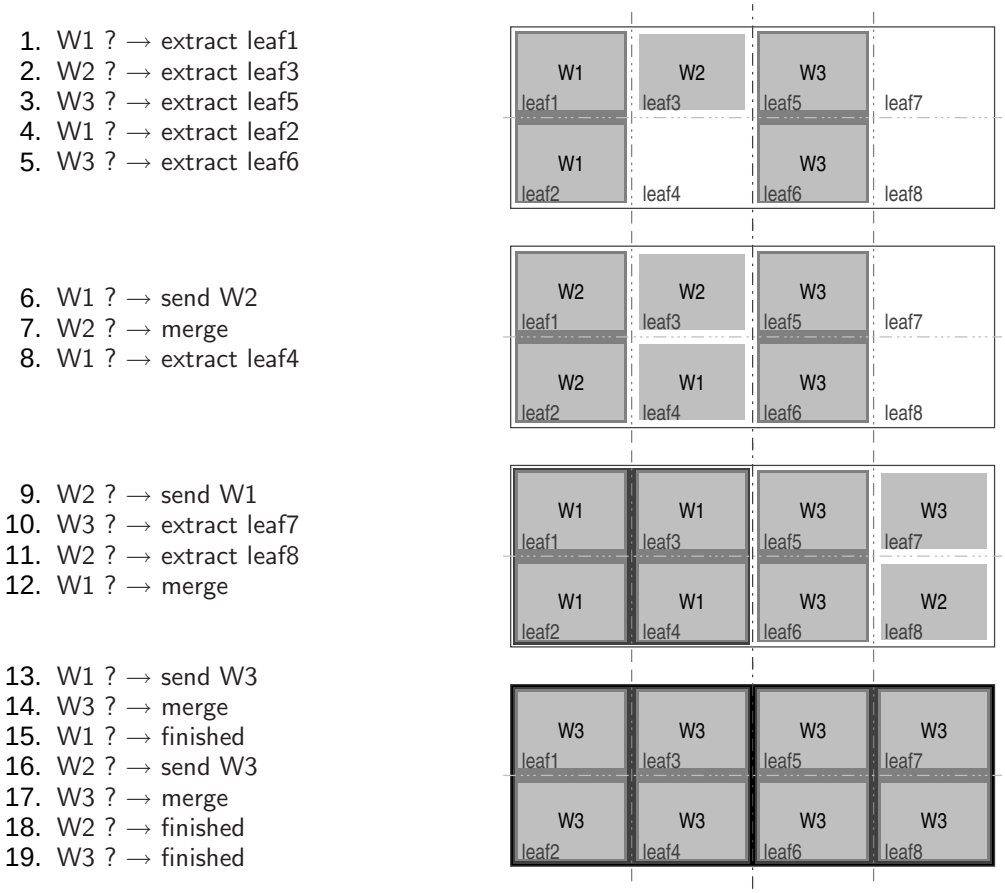


Fig. 10. Algorithm trace that could be obtained with three workers on a dataset recursively subdivided in eight blocks. Grayed blocks depict processed ones and the worker that own them currently is indicated by its id (W1 to W3). Each of the four states is the result of the block of tasks written to its left.

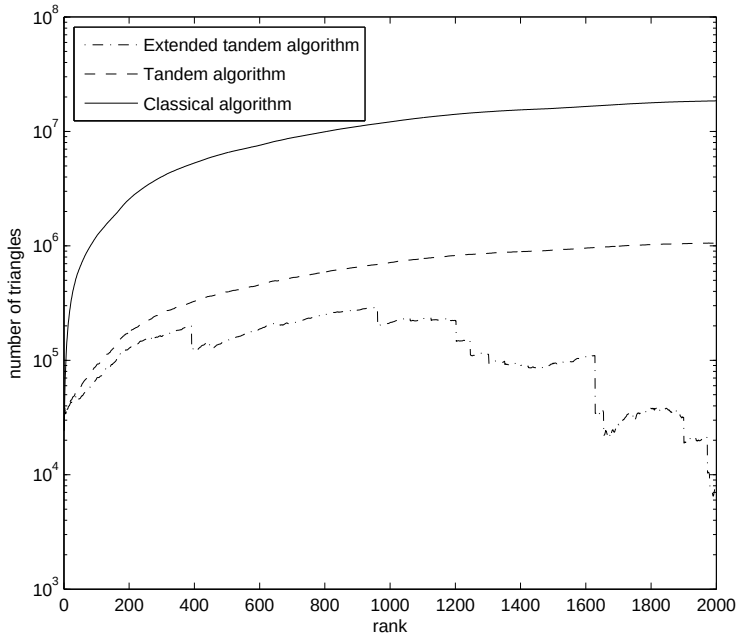


Fig. 11. Evolving size of the triangulated surface for three algorithms: the classical marching-cubes algorithm (extraction of the entire surface and simplification), the tandem algorithm and our tandem algorithm with dumping of finished components. Computations are performed on a cube of size $401 \times 1051 \times 2001$.

4 Computational experiments

In order to analyze the performance of our algorithm, we focus on memory consumption during the extraction and on analyzing the quality of the generated mesh.

Memory consumption A critical point with the marching-cubes algorithm is the amount of memory needed to store the generated triangles. Figure 11 illustrates the comparison between a brute-force approach, the tandem algorithm and our tandem algorithm with dumping. Triangles are counted right after processing of a layer k. We see that the tandem algorithm drastically reduces the number of triangles generated compared to the classical approach. Our dumping strategy enhances the tandem algorithm by clearly reducing further the number of triangles.

Mesh quality Many applications need "well-shaped" triangles instead of long and skinny ones. A classical evaluation of mesh quality is to measure the aspect ratio of a triangle $t = abc$ as $\rho(t) = \sqrt{\lambda_2/\lambda_1}$ where $\lambda_1 \geq \lambda_2$ are the

greatest eigenvalues of the inertia matrix of t, defined by:

$$M_t = \frac{1}{3}[(a-\hat{t})(a-\hat{t})^T + (b-\hat{t})(b-\hat{t})^T + (c-\hat{t})(c-\hat{t})^T]$$

with $\hat{t}$ the centroid of triangle t. $\rho(t) = 1$ if the triangle t is equilateral ($\lambda_1 = \lambda_2$) and $\rho(t) = 0$ if it is flat ($\lambda_2 = 0$). As a global measure of the triangulation K (composed by n triangles), Attali et al. [1] use a metrics of the anisotropy, also called "skewness" in the mesh generation field:

$$anisotropy(K) = 1 - \frac{1}{n}\sum_t \rho(t)$$

with $0 \leq anisotropy(K) \leq 1$ and $anisotropy(K) = 0$ if all triangles are equilateral.

Figure 12 shows anisotropy variation as a function of the isotropy parameter α for the three approaches: the sequential tandem algorithm and the parallel algorithm with and without time lag. We clearly see the effect of the time lag on the quality of the generated mesh. Parallel extraction generates a mesh with a quality equivalent to that of the sequential approach. The increase of anisotropy (α greater than 0.9) is due to the fact that the shape criterion is no more taken into account in the choice of candidates for edge collapse and therefore the collapse operation is only based on triangle shape quality, and proposed points far from the original surface: most of these candidates are

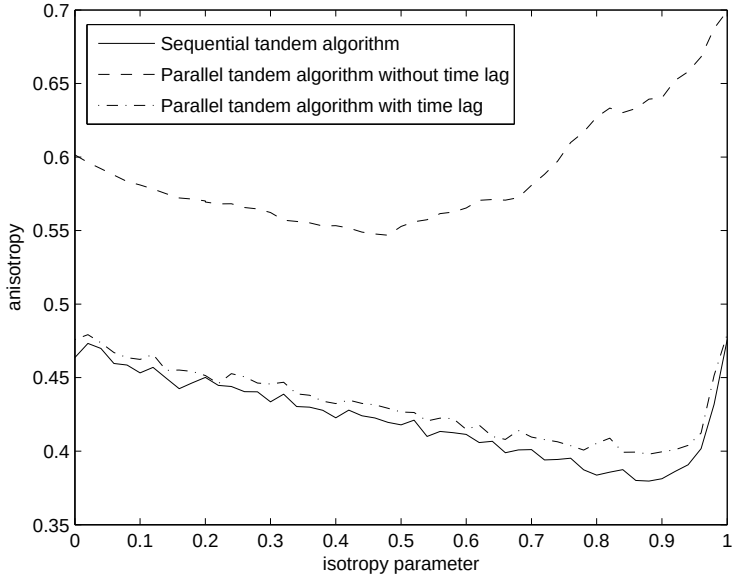


Fig. 12. Anisotropy variation of the mesh as a function of the isotropy parameter α. Thank to the extension of the time lag concept to the block boundaries, our parallel algorithm gives as good mesh quality measurements as with the original sequential version of the tandem algorithm.

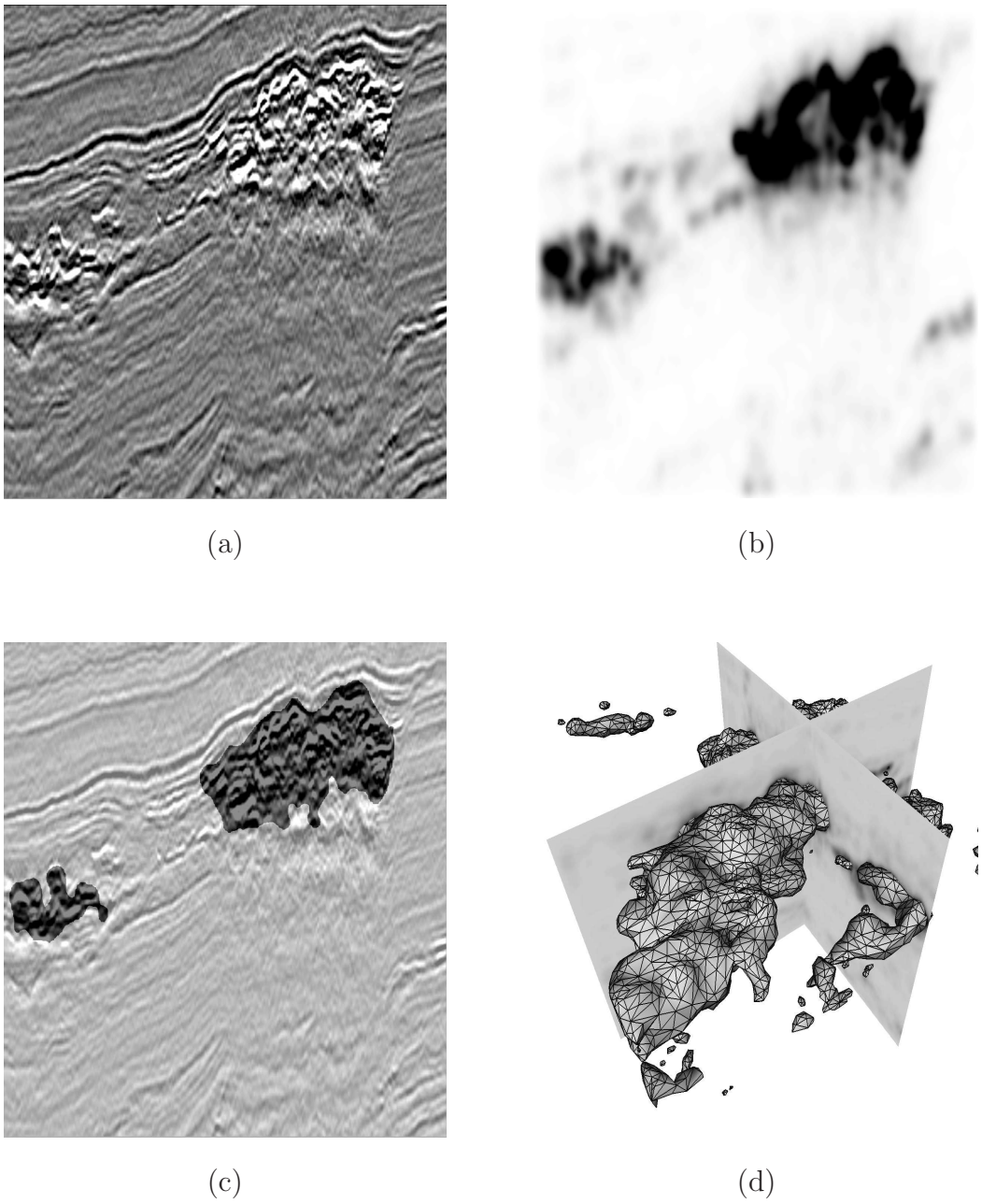
rejected by the error threshold E_0. In practice, the cost function must always take care of the shape measure and then prevent these degenerated cases: a value of α equal to 0.4 is a good compromise.

Application to a geophysical example Seismic cubes are very common datasets used in hydrocarbon exploration and analysis of such datasets is very helpful to characterize depositional environment. A common tool for seismic analysis is based on texture analysis and seismic attributes computation. A seismic attribute highlights specific features of the dataset. For example Figure 13 shows an example of attribute that detects the chaotic and high-amplitude areas in the seismic dataset. As such facies could be related to the presence of hydrocarbons, making an inventory of all these areas is very helpful for geophysicists. In order to make this inventory, we apply the following workflow: attribute computation, surface extraction and disconnected components sorting. As the attribute highlights the interesting parts, by adjusting a threshold we extract the 3D shapes with our proposed approach (figure 13).

This workflow is applied to a dataset of size 1626×7028×2000, i.e. 40 gigabytes. The first interest of our approach is its parallelism: according to the number of available processors, we reduce the extraction time (the extraction is performed in 5 minutes with 56 processors instead of more than three hours for the tandem algorithm). Figure 14 shows all the extracted surfaces, and one can clearly see many small extracted objects that are probably due to noise in the attribute or to very small objects. In order to make a ranking of all these surfaces, we can easily sort them according to geometrical criteria. Figure 15 illustrates this sorting by eliminating the smallest objects. Only 328 components are kept over the 18 111 in the unfiltered version. This workflow is helpful to locate such areas and to analyze their distribution and relation.

5 Conclusion

Due to the increasing size of datasets, the literature on isosurfaces extraction has focused in recent years on approaches that extract simplified surfaces or dump surfaces as soup of triangles. But none of these approaches propose parallel extraction with simplification and dumping of disconnected components. Our component-based dumping approach implies a file organization allowing interactive exploration of the volume. This organization could be applied to other surface reconstruction algorithms that generate many disconnected components. Our parallel processing based on dataset splitting and time lag extension could also be beneficial to simplification methods that split their surfaces into patches.



(a) (b)

(c) (d)

Fig. 13. A typical seismic attribute. (a) Cross sectional view of a seismic dataset. (b) Cross sectional view of an attribute computed on the seismic dataset. (c) Superposition of the attribute (the color palette is saturated at a given threshold) over the seismic data. (d) Global view of the extracted surfaces from the seismic attribute volume.

Acknowledgments

This work was supported by the TOTAL company under CIFRE grant number 649/2005.

References

[1] D. Attali, D. Cohen-Steiner, H. Edelsbrunner, Extraction and simplification of iso-surfaces in tandem, in: SGP '05: Proceedings of the third Eurographics symposium on Geometry processing, Eurographics Association, Aire-la-Ville, Switzerland, 2005, pp. 139–148.

[2] W. E. Lorensen, H. E. Cline, Marching cubes: A high-resolution 3D surface construction algorithm, in: SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM Press, New York, NY, USA, 1987, pp. 163–169.

[3] A. V. Gelder, J. Wilhelms, Topological considerations in isosurface generation, ACM Transactions on Graphics 13 (4) (1994) 337–375.
URL `citeseer.ist.psu.edu/gelder94topological.html`

[4] J. Wilhelms, A. V. Gelder, Octrees for faster isosurface generation, ACM Transactions on Graphics 11 (3) (1992) 201–227.

[5] Y. Livnat, H.-W. Shen, C. R. Johnson, A near optimal isosurface extraction algorithm using the span space, IEEE Transactions on Visualization and Computer Graphics 2 (1) (1996) 73–84.
URL `citeseer.ist.psu.edu/livnat96near.html`

[6] S. Niguet, J.-M. Nicod, A load-balanced parallel implementation of the marching-cubes algorithm, Tech. Rep. 95-24, Laboratoire de l'informatique du parallélisme de l'École Normale Supérieure de Lyon, France (1995).

[7] P. Mackerras, A fast parallel marching-cubes implementation on the Fujitsu AP1000, Tech. Rep. TR-CS-92-10, Canberra 0200 ACT, Australia (1992).
URL `citeseer.ist.psu.edu/mackerras92fast.html`

[8] R. Shekhar, E. Fayyad, R. Yagel, J. F. Cornhill, Octree-based decimation of marching cubes surfaces, in: IEEE Visualization, 1996, pp. 335–342.
URL `citeseer.ist.psu.edu/shekhar96octreebased.html`

[9] P. Lindstrom, Out-of-core simplification of large polygonal models, in: SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000, pp. 259–262.

[10] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, External memory management and simplification of huge meshes, IEEE Transactions on Visualization and Computer Graphics 9 (4) (2003) 525–537.

[11] J. Wu, L. P. Kobbelt, A stream algorithm for the decimation of massive meshes, in: Graphics Interface 2003 Proceedings, 2003, pp. 185–192.
URL `www.graphicsinterface.org/proceedings/2003/118/file118-1.pdf`

[12] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, Computer Graphics 31 (Annual Conference Series) (1997) 209–216.
URL `citeseer.ist.psu.edu/garland97surface.html`

[13] F. Pivot, G. Dupuy, S. Guillon, J. N. Ferry, Complex volumic seismic interpretation by new geobodies extraction strategy, in: London 2007, 69th EAGE Conference & Exhibition incorporating SPE EUROPEC 2007, 2007.

[14] H. Muller, M. Stark, Adaptive generation of surfaces in volume data (1993). URL citeseer.ist.psu.edu/article/muller91adaptive.html

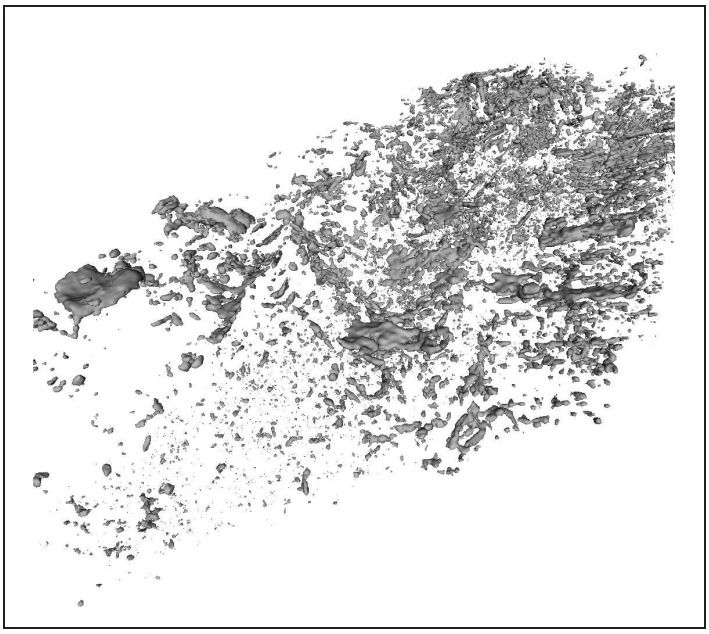


Fig. 14. Example of a surface generated for a dataset of size 1626×7028×2000. This surface contains 18 111 disconnected components.

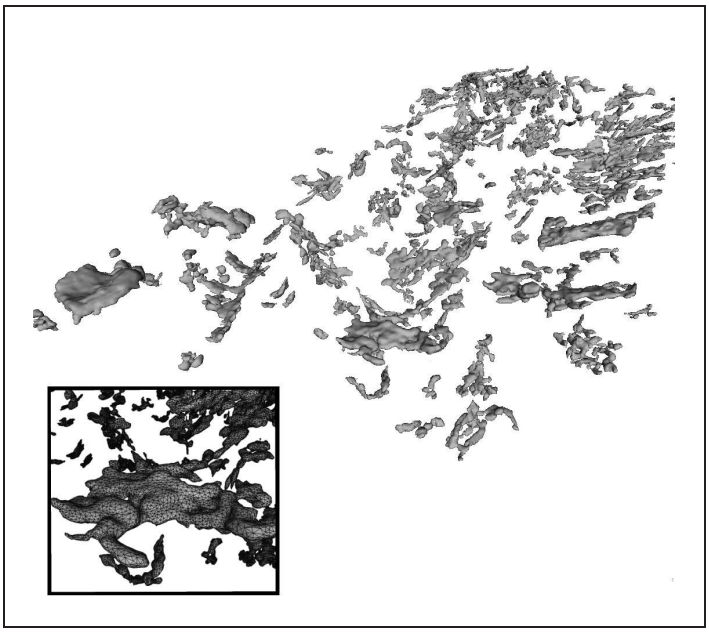


Fig. 15. Components extracted on a dataset of size 1626×7028×2000 (see figure 14) and filtered according to their volumes. This surface contains 328 disconnected components.