Efficient geometric reconstruction of complex geological structures

Dassi, F.; Perotto, S.; Formaggia, L.; Ruffo, P.

MOX, Dipartimento di Matematica "F. Brioschi"
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox@mate.polimi.it http://mox.polimi.it

Efficient geometric reconstruction of complex geological structures

F. Dassi[1], S. Perotto[1], L. Formaggia[1] and P. Ruffo[2]

October 23, 2012

[1]MOX, Politecnico di Milano, Dipartimento di Matematica “F. Brioschi”
Piazza Leonardo da Vinci 32, I-20133, Milano, ITALY
{franco.dassi,simona.perotto,luca.formaggia}@polimi.it
[2] ENI - Exploration & Production Division, GEBA Dept.
Via Emilia 1, I-20097, San Donato Milanese, ITALY
Paolo.Ruffo@eni.com

23-10-2012

Keywords: triangulated surface intersection, search data structures, geological model reconstruction.

AMS Subject Classification: 51 65 90

Abstract

Complex geological structures pose a challenge to domain discretization. Indeed data are normally given as a set of intersecting surfaces, sometimes with incomplete data, from which one has to identify the computational domain to build a mesh suited for numerical simulations. In this paper we describe a set of tools which have been developed for this purpose. Specialized data structures have been developed to efficiently identify intersections of triangulated surfaces and to conformally include these intersections in the starting meshes, while improving the mesh quality. Then, an effective algorithm has been implemented to detect the different sub-regions forming the computational domain; this algorithm has been properly enhanced to take into account the specific characteristics involved in the simulation of geological basins.

1 Introduction

In many applications the geometry of the computational domain comes from different kind of tomography processes, acoustic, seismic and X-ray imaging being only few examples. In fact, these processes are usually able to give information on the geometry of

the surfaces that bound the different parts of the domain of interest. Specific procedures are then necessary to convert this information to a geometrical description suitable for the successive analyses, which typically involve the numerical solution of partial differential equations.
For example, data from seismic imaging offer a description of the horizons, i.e., a set of three dimensional surfaces that represent the deposition of different kind of sediments (see Figure 1, left). Moving from these data, we have to get a volume discretization representing the whole sedimentary basin (see Figure 1, right).

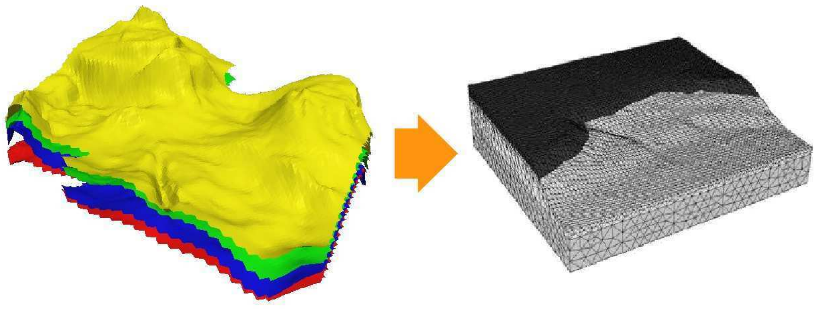

Figure 1: Surfaces that represent the horizons (left); the volume of interest (right).

We are indeed interested in the geometrical reconstruction of a sedimentary basin moving from the description of horizons, usually given in the form of a triangulated surfaces There is a sequence of operations that are required to finally obtain a computational mesh representing the whole sedimentary basin on which one can carry out numerical simulations. We will focus on the following procedures:

a) identification of surface intersection;

b) detection of regions enclosed by this intersection and generation of a conformal mesh which includes such intersection;

c) improvement of the mesh quality.

In geological applications, the data sets that represent the different horizons have large size; so an efficient implementation of the previous algorithms, especially of the surface intersection, becomes important. We have devised proper data structures based on the combination of an Alternated Binary Tree with a Structured Space discretization, to address this issue and to obtain an efficient code.
The intersections partition the surface into regions which have to be identified, since they represent the boundary of different portions of the computational domain, typically with different characteristics. We propose here a simple algorithm for the automatic recognition of those regions. In this process we also remesh the surface so that the new triangulation is conformal with the intersections. This process is complemented by a mesh enhancement procedure to avoid badly shaped triangles. Finally, we are able to

identify the different portions of the three dimensional geometry automatically.
Another aspect peculiar to the target application is that data are often incomplete. We thus need to reconstruct the missing parts. We present here two possible strategies that we have successfully implemented.

The paper is organized as follows. Section 2 deals with the problem of finding the intersection between two triangulated surfaces and details the data structures that have been used for this purpose. The localization of the intersection is a prerequisite for the procedure we illustrate in Section 3, where we deal with the automatic detection of the different regions composing our 3D model. The surface mesh provided by the original data set and successively modified by the intersection finding procedure is often not suitable for computations. The quality of the elements is usually poor. In Section 4 we outline the mesh improvement strategies we have consequently adopted. In Section 5 we describe some techniques specific for the application at hand, namely the completion of defective geometrical data, the treatment of the so-called hard and soft horizons, and the construction of sub-volumes. Finally, Section 6 provides some more quantitative results about the algorithms proposed in Sections 2–3. Some conclusions are drawn in the last section.

2 Intersection of horizons and faults

The goal of this section is to describe an efficient method to detect the intersection between horizons and faults in a sedimentary basin. The latter are described by triangulated surfaces.
For the sake of simplicity, we focus on a single intersection between a fault and a horizon, even though realistic geological configurations include multiple intersections (see Figure 2 for two examples). We can consequently formalize the detection of such an intersection via the simple geometric configuration illustrated in Figure 3. Here, the blue surface $\mathcal{S}^B$ represents a fault which intersects the horizon given by the red surface $\mathcal{S}^R$. In particular, since we deal with two triangulated surfaces, we are led to identify the yellow piecewise linear curve Γ in Figure 3, right, i.e., to search the intersection between couples of non-coplanar triangles. To get an efficient procedure, we need a fast algorithm to identify the couples of triangles intersecting each other. The most straightforward approach, which consists in checking the intersection, for instance, of each triangle of $\mathcal{S}^B$ with each triangle of $\mathcal{S}^R$, is unavoidably ineffective when dealing with large data sets.
A more efficient criterion consists in finding a fast procedure to select, for each triangle $T \in \mathcal{S}^B$, a subset $\mathcal{D}_T \subset \mathcal{S}^R$ of triangles surrouding T and containing the possible intersecting triangle(s) (see Figure 4). We can then check the actual intersection only on the elements in $\mathcal{D}_T$.

The availability of a proper search data structure becomes crucial for a quick detection of $\mathcal{D}_T$, which should enjoy the following properties:

i) to contain the intersection $T \cap \mathcal{S}^R$;

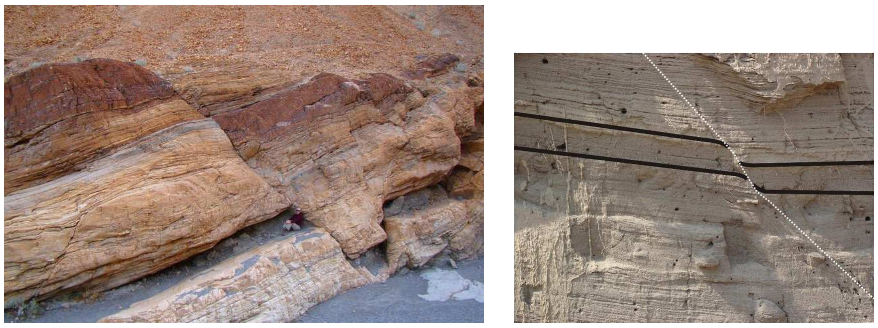
Figure 2: Examples of geological configurations where horizons are crossed by faults.

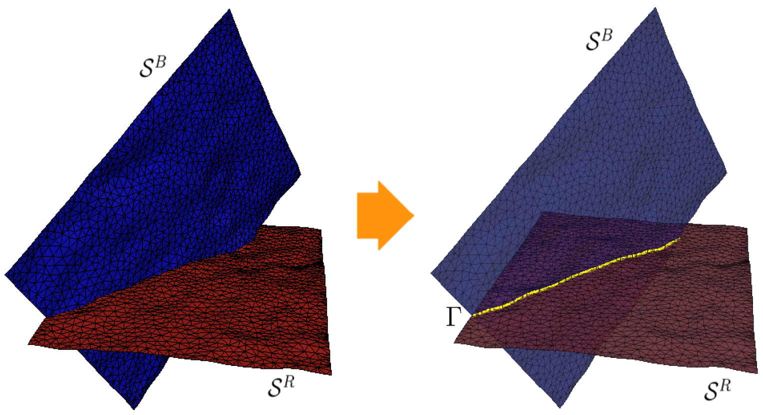

Figure 3: Non-coplanar surfaces (left) and corresponding intersection (the yellow line, on the right).

ii) to be as small as possible;

iii) to be found rapidly.

In Section 2.3 we propose an intersection algorithm particularly suited to deal with the geological configurations we are interested in. It essentially merges two intersection procedures based on well-established data structures which we summarize in Sections 2.1 and 2.2, respectively. Independently of the selected approach, the idea is first to build a suitable data structure associated with one of the two surfaces and then to look for the intersection with the other surface.

2.1 An intersection algorithm based on a structured data search

We refer to the approach proposed, e.g., in [6]. We associate the data structure with the horizon $\mathcal{S}^R$. For this purpose, we denote by $\mathcal{N}^R$ and $\mathcal{T}^R$ the set of the nodes and of the

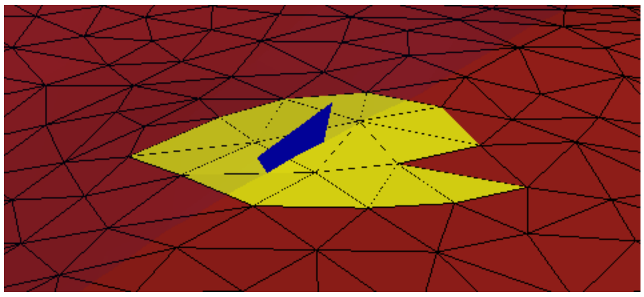

Figure 4: Set $\mathcal{D}_T$ (the yellow area) associated with the triangle $T \in \mathcal{S}^B$.

elements of the triangulated surface $\mathcal{S}^R$, respectively, and we indicate the generic 3D coordinate system by $(0, x, y, z)$. We build the *bounding box* $\mathcal{B}(\mathcal{S}^R)$ associated with $\mathcal{S}^R$, i.e., the smallest box containing the whole surface $\mathcal{S}^R$, identified by the two points $SW^R = (x_{sw}, y_{sw}, z_{sw})$ and $NE^R = (x_{ne}, y_{ne}, z_{ne})$, of coordinates

$$x_{sw} = \min_{P\in\mathcal{N}^R} x_p, \quad y_{sw} = \min_{P\in\mathcal{N}^R} y_p, \quad z_{sw} = \min_{P\in\mathcal{N}^R} z_p, \tag{1}$$

$$x_{ne} = \max_{P\in\mathcal{N}^R} x_p, \quad y_{ne} = \max_{P\in\mathcal{N}^R} y_p, \quad z_{ne} = \max_{P\in\mathcal{N}^R} z_p, \tag{2}$$

where $P = (x_p, y_p, z_p)$ is the generic node of $\mathcal{S}^R$ (see Figure 5, left for an example). We successively subdivide $\mathcal{B}(\mathcal{S}^R)$ into fixed-size sub-boxes, of dimensions

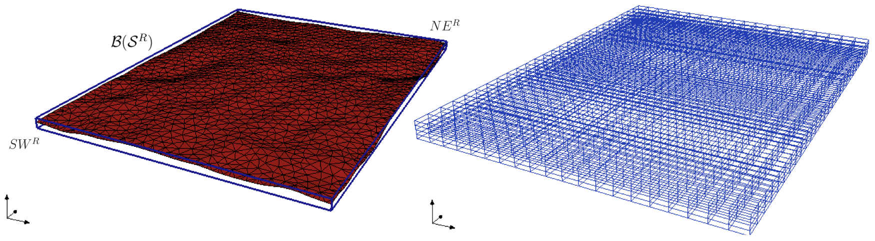

Figure 5: Bounding box (left) and corresponding hexahedral mesh (right) for the surface $\mathcal{S}^R$.

$$L_x = \max_{K\in\mathcal{T}^R} dx_K, \quad L_y = \max_{K\in\mathcal{T}^R} dy_K, \quad L_z = \max_{K\in\mathcal{T}^R} dz_K, \tag{3}$$

where

$$dx_K = \max_{Q,R\in\mathcal{N}_K} |x_q - x_r|, \quad dy_K = \max_{Q,R\in\mathcal{N}_K} |y_q - y_r|, \quad dz_K = \max_{Q,R\in\mathcal{N}_K} |z_q - z_r|, \tag{4}$$

are the dimensions of the bounding box $\mathcal{B}(K)$ associated with the generic triangle $K \in \mathcal{T}^R$ (see Figure 7, left), while $Q = (x_q, y_q, z_q)$ and $R = (x_r, y_r, z_r)$ are chosen in the

set $\mathcal{N}_K$ of the nodes of K.
The 3D space is consequently organized into a *structured cartesian* hexahedral mesh (see Figure 5, right). This spatial discretization is very flexible: it can be employed to organize any n-dimensional space after properly redefining the involved geometrical elements. It is also suited to parallelization since the hexahedral mesh can be subdivided into sub-blocks distributed among processors. Moreover, it allows us to identify the cell containing a certain point $P \in \mathbb{R}^n$ in a fast way. Let us exemplify this property in a 2D setting. As shown in Figure 6, the bounding box now coincides with the rectangle defined by the points $SW^R = (x_{sw}, y_{sw})$ and $NE^R = (x_{ne}, y_{ne})$, and it is subdivided, for instance, into $N_x(= 5)$ and $N_y(= 4)$ cells of lenght L_x and L_y along the x-axis and y-axis, respectively. The numbering of the cells follows a lexicographic order. This allows us to immediately find the identificator Id_P of the cell containing the point $P = (x_p, y_p)$, via the formula

$$\text{Id}_P = X_{\text{Id}} + Y_{\text{Id}} N_x, \tag{5}$$

where

$$X_{\text{Id}} = \left\lfloor \frac{x_p - x_{sw}}{L_x} \right\rfloor, \quad Y_{\text{Id}} = \left\lfloor \frac{y_p - y_{sw}}{L_y} \right\rfloor, \tag{6}$$

and where $\lfloor \cdot \rfloor$ denotes the standard floor function. Formula (5) generalizes to

$$\text{Id}_P = X_{\text{Id}} + Y_{\text{Id}} N_x + Z_{\text{Id}} N_x N_y \tag{7}$$

in the 3D case, with

$$X_{\text{Id}} = \left\lfloor \frac{x_p - x_{sw}}{L_x} \right\rfloor, \quad Y_{\text{Id}} = \left\lfloor \frac{y_p - y_{sw}}{L_y} \right\rfloor, \quad Z_{\text{Id}} = \left\lfloor \frac{z_p - z_{sw}}{L_z} \right\rfloor, \tag{8}$$

$P = (x_p, y_p, z_p)$ and L_z the size of the sub-boxes of $\mathcal{B}(\mathcal{S}^R)$ along the z-axis.

We now store all the triangles in $\mathcal{T}^R$ into the hexahedral mesh defined by (3)-(4): a priori, each sub-box may contain any number of elements, including none (see Figure 7, right for an example, where different elements share the same hexahedral box). For any triangle $K \in \mathcal{T}^R$, we introduce the center $C_K = (x_{C_K}, y_{C_K}, z_{C_K})$ of the bounding box $\mathcal{B}(K)$ as representative of K, where

$$x_{C_K} = \frac{x_{sw}^K + x_{ne}^K}{2}, \quad y_{C_K} = \frac{y_{sw}^K + y_{ne}^K}{2}, \quad z_{C_K} = \frac{z_{sw}^K + z_{ne}^K}{2},$$

while $SW^K = (x_{sw}^K, y_{sw}^K, z_{sw}^K)$ and $NE^K = (x_{ne}^K, y_{ne}^K, z_{ne}^K)$ are the two points identifying $\mathcal{B}(K)$, with cartesian coordinates defined according to (1)-(2), for $P \in \mathcal{N}_K$ (see Figure 7, left). Then, the triangle K is stored in the hexahedral cell where its representative C_K falls, according to criterion (7). At this point it is rather easy to detect all the triangles close to a certain point P, since they are stored in the hexahedral cell containing P, whose identificator is given by (7).

Now, by exploting this data structure, it is possible to construct, for each element $T \in \mathcal{S}^B$, the set $\mathcal{D}_T \subset \mathcal{S}^R$ of the triangles in $\mathcal{S}^R$ close to T ([6]). We consider the

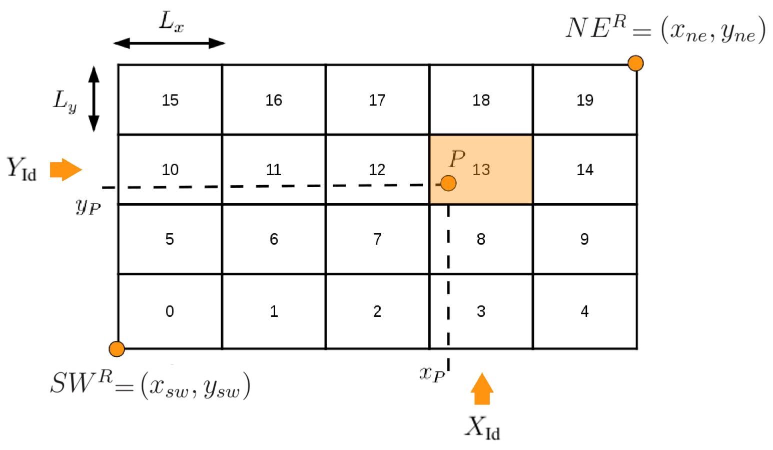

Figure 6: Example of direct addressing associated with structured data in a 2D case.

bounding box $\mathcal{B}(T)$; then, we build the set $\mathcal{H}_T$ of the hexahedral cells of $\mathcal{S}^R$ intersecting $\mathcal{B}(T)$ moving from the identificators Id_{SW^T} and Id_{NE^T} of the hexahedral cells of $\mathcal{S}^R$ containing SW^T and NE^T. The set $\mathcal{D}_T$ is thus defined by the union of all the triangles contained in $\mathcal{H}_T$. Finally, the segment representing the intersection $T \cap \mathcal{D}_T$ is found by resorting to a standard triangle-triangle *intersection test*; in particular, following [1], we have adopted the one proposed in [5].

We remark that the set $\mathcal{D}_T$ does not satisfy the requirement of minimal possible extension since it may include triangles that do not actually intersect the bounding box $\mathcal{B}(T)$ (we refer to Figure 8 for an example of non optimal detection of $\mathcal{D}_T$). Useless triangle-triangle intersection tests are consequently performed. The algorithm can be improved by performing a bounding-box intersection test before checking the actual triangle-triangle intersection.

2.2 An intersection algorithm based on an AB search

We present here an alternative data structure for the horizon $\mathcal{S}^R$, based on a binary tree. More precisely, we resort to an alternate binary (AB) search tree ([6]) (see Figure 9 for an example).

Let us explain how an element $K \in \mathcal{S}^R$ is stored in an AB data structure. At the beginning of the storage procedure, the tree data structure is empty, so that the triangle first considered is stored in the root ①. Afterwards, each element K is stored, starting from the root, in the first available node identified via a precise criterium.
Namely, we consider the vector $\vec{v}_K = (x^K_{sw}, x^K_{ne}, y^K_{sw}, y^K_{ne}, z^K_{sw}, z^K_{ne}) \in \mathbb{R}^6$ associated with the bounding box $\mathcal{B}(K)$ of the element K, i.e., collecting the coordinates of the points $SW^K = (x^K_{sw}, y^K_{sw}, z^K_{sw})$ and $NE^K = (x^K_{ne}, y^K_{ne}, z^K_{ne})$ (see Figure 7, left), as well as the vector $\vec{v}_\tau = (x^\tau_{sw}, x^\tau_{ne}, y^\tau_{sw}, y^\tau_{ne}, z^\tau_{sw}, z^\tau_{ne})$ associated with the triangle stored at

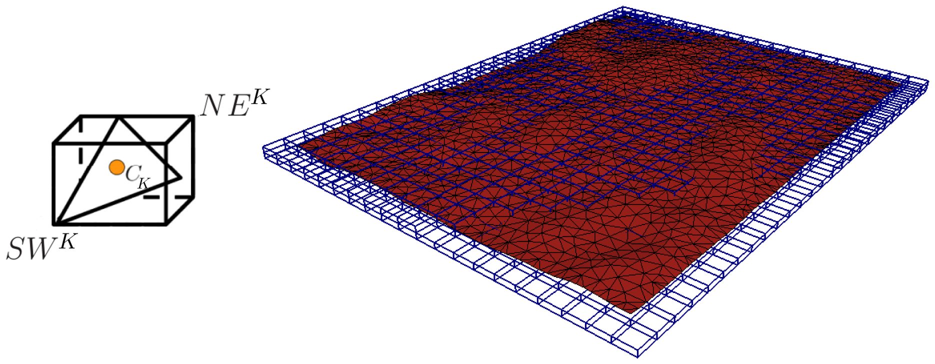

Figure 7: Bounding box $\mathcal{B}(K)$with the representative C_K (left); surface $\mathcal{S}^R$ stored in the corresponding hexahedral mesh (right).

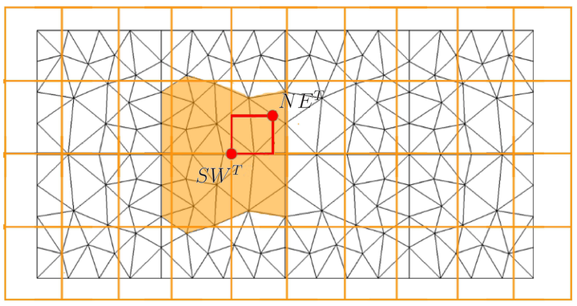

Figure 8: Detection of the set $\mathcal{D}_T$ (the orange area) via the structured-data approach.

the generic node τ of the tree. Triangle K finds a location in the tree by properly comparing vectors $\vec{v}_K$ and $\vec{v}_\tau$. In particular, let $\mathtt{j}_\tau$ denote the depth of the node τ and let $v_{K,i}$ be the i-th component of the vector $\vec{v}_K$. Then, we apply the following algorithm:

```
$\tau$ $\leftarrow$ root; j$_\tau$ = 0;
if $\tau$ is empty,
   $\tau \leftarrow K$, return;
$\alpha$ = j$_\tau$(mod 6);
if $v_{K,\alpha}$ < $v_{\tau,\alpha}$,
   $\tau$ $\leftarrow$ $\tau$.left;
else
   $\tau$ $\leftarrow$ $\tau$.right;
end
```

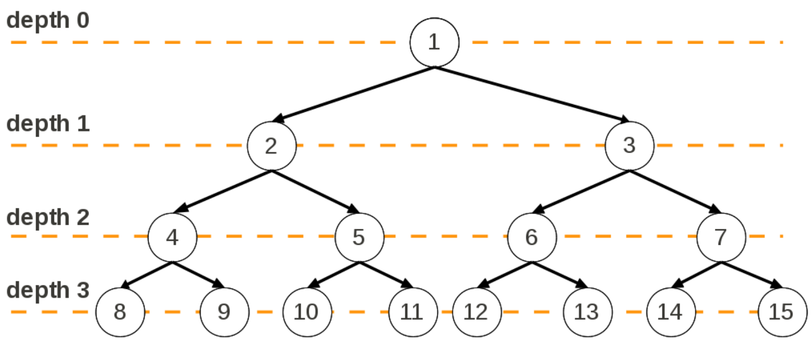

Figure 9: Example of AB search tree.

```
j$_\tau$=j$_\tau$+1;
```

where τ.left and τ.right is the child at the left and at the right of τ, respectively. The comparison test in the algorithm depends on the depth of the tree: the inequalities *alternate* both the extremes of the bounding boxes and the coordinates, x, y and z. This justifies the name of the algorithm as well as it makes this approach suited to organize data in any dimension.
Once an AB data structure is built for the horizon $\mathcal{S}^R$, we proceed with the intersection with the fault $\mathcal{S}^B$. For any element $T \in \mathcal{S}^B$, we find the set $\mathcal{D}_T$ of the triangles in $\mathcal{S}^R$ near T. In particular, we build the bounding box $\mathcal{B}(T)$ associated with T; we go through the binary tree of $\mathcal{S}^R$, moving from the root. The node τ of the tree is included in $\mathcal{D}_T$ if the corresponding bounding box $\mathcal{B}(\tau)$ intersects $\mathcal{B}(T)$. Then, we proceed recursively by examining the children of τ. We can summarize this procedure via the algorithm below:
let j$_\tau$ and $\vec{v}_K$ be defined as above; let S and Q be two stacks[1] initially empty (S is a temporary stack, while Q will contain the list of the triangles constituting $\mathcal{D}_T$). Then

```
S.push(root)

Until S is empty
        $\tau$ = S.pop;
        j$_\tau$ = depth($\tau$);
        $\alpha$ = j$_\tau$(mod 6);

        if $\alpha$ is even,
            k = $\alpha$ + 1;
            if $v_{T,\mathrm{k}}$ < $v_{\tau,\alpha}$,
                S.push($\tau$.left);
```

[1]We remind that a stack is characterized by two standard operations, called push and pop: the push operation adds a new item to the top of the stack; viceversa, the pop operation removes an item from the top of the stack

```
            else
                Q.push($\tau$);
                S.push($\tau$.left);
                S.push($\tau$.right);
            end
        else
            k = $\alpha$ - 1;
            if $v_{T,\mathtt{k}} \geq v_{\tau,\alpha}$,
                S.push($\tau$.right);
            else
                Q.push($\tau$);
                S.push($\tau$.left);
                S.push($\tau$.right);
            end
        end
end
```

The region $\mathcal{D}_T$ identified via an AB data structure is, in general, different than the one obtained via a structured data search. The AB approach allows us to build a really confined set $\mathcal{D}_T$, where only the triangles of $\mathcal{S}^R$ whose bounding box actually intersects $\mathcal{B}(T)$ are included. This property is not a priori guaranteed by the structured-data based approach, as shown in Figure 8.

Moreover, at each step of the procedure, one of the two children of τ is excluded from the search. To maximize the benefits due to this "cutting of branches", we should have a tree as balanced as possible (like the one in Figure 9). To exemplify the negative effect of unbalancing, let us consider the AB data structure associated with the mesh in Figure 10. If we assign the triangle number 1 to the root, we get a tree completely unbalanced since all the elements of the mesh are stored at the right of the root. No elimination criterion is consequently exploited at depth 0.

It is possible to extend the data structure to achieve automatically a good balancing (see [6]); yet in our case a good balancing may be obtained by suitably modifying the strategy for the insertion of the elements in the tree structure. We have chosen the following approach: for a certain surface $\mathcal{S}$, we first define the six vectors SW_x, SW_y, SW_z, NE_x, NE_y, NE_z that collect, for each element $K \in \mathcal{S}$, the coordinates x, y, z of the points SW^K and NE^K defining the bounding box $\mathcal{B}(K)$. We sort the elements in each vector into the ascending order. Then, we build the binary tree data structure following a dichotomic principle. At depth 0, as root of the tree, we select the triangle coinciding with the median of the vector SW_x; the two nodes at depth 1 are represented by the two 4-quantiles of the vector SW_y of order $1/4$ and $3/4$; the four nodes at depth 2 are provided by the four 8-quantiles of the vector SW_z of order $1/8$, $3/8$, $5/8$, $7/8$, and so on. Table 1 formalizes this idea level by level, where $\mathcal{I}_t = \{\frac{2j+1}{t} < 1, j \in \mathbb{N}\}$, with $t = 2^{\mathtt{i}+1}$ and i the considered depth of the tree. Of course, the triangle K is removed from the six vectors as soon as it is stored in the binary tree. With reference to

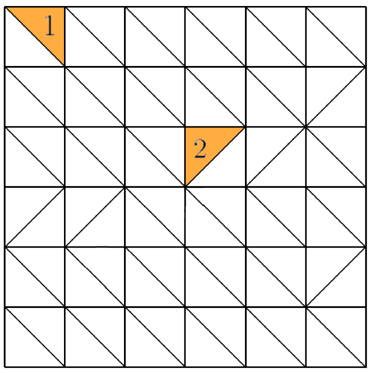

Figure 10: A 2D planar surface $\mathcal{S}$: two possible choices for the root, leading to a completely unbalanced (triangle 1) and to a balanced (triangle 2) binary tree data structure.

Figure 10, the choice, e.g., of the triangle 2 as root leads to a balanced binary tree.

2.3 Coupling structured and AB data search

The two algorithms in Sections 2.1 and 2.2 have complementary characteristics with respect to properties i)-iii) itemized at the beginning of this section. Both the approaches guarantee the first property. The algorithm based on a structured data search detects the set $\mathcal{D}_T$ very rapidly, in practice each search has $O(1)$ complexity. But this set can be rather large, thus violating requirement ii). On the contrary, the algorithm based on the AB search builds a set $\mathcal{D}_T$ with a rather small number of elements. However, this procedure is, in general, less efficient, the complexity of a single search being (for a balanced tree) of order $O(logN)$, with N the number of elements in the tree.

Our actual goal is to combine these two procedures into a new algorithm able to merge the respective advantages. The basic idea consists of reducing the size of the problem $\mathcal{S}^R \cap \mathcal{S}^B$, by confining the intersection to suitable subsets $\mathcal{S}_s^R$ and $\mathcal{S}_s^B$ of $\mathcal{S}^R$ and $\mathcal{S}^B$, respectively (see Figure 11 for a sketch of the procedure). In more detail, the reduction step, that is the generation of the sub-meshes $\mathcal{S}_s^R$ and $\mathcal{S}_s^B$, is performed via the quick (but rough) algorithm; the intersection phase is assigned to the sharp AB-algorithm: the slower speed of the AB approach is balanced by the reduced number of mesh elements now involved in the intersection. We can think about a sort of predictor-corrector approach, where the structured data search is the predictor and the AB-algorithm is the corrector.

To itemize the main steps of the proposed procedure, we refer to the geometric configuration in Figure 11 (the same as in Figure 3).

1. we associate a *unique* data structure to the union $\mathcal{S}^R \cup \mathcal{S}^B$ of the two intersecting surfaces, following the approach in Section 2.1: the hexahedral mesh now contains

depth of the tree	reference vector	quantile orders
0	SW_x	$\mathcal{I}_2$
1	SW_y	$\mathcal{I}_4$
2	SW_z	$\mathcal{I}_8$
3	NE_x	$\mathcal{I}_{16}$
4	NE_y	$\mathcal{I}_{32}$
5	NE_z	$\mathcal{I}_{64}$
6	SW_x	$\mathcal{I}_{128}$
7	SW_y	$\mathcal{I}_{256}$
$\vdots$	$\vdots$	$\vdots$

Table 1: Vectors and quantile orders associated with the different depths of the tree to build a balanced binary tree.

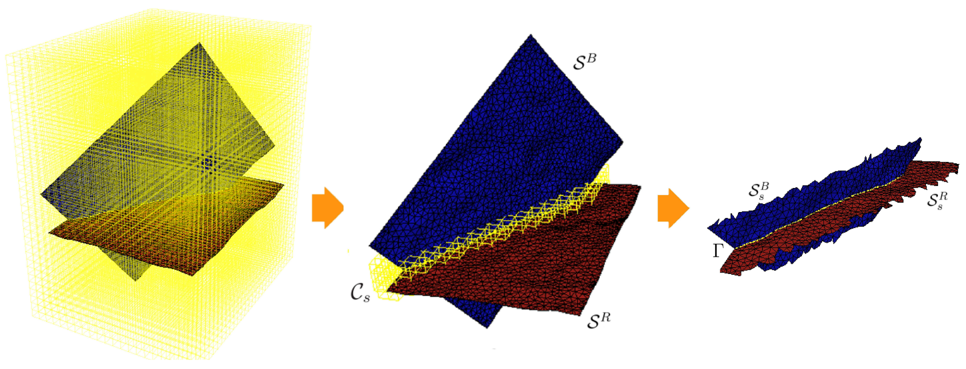

Figure 11: Reduction of the intersection $\mathcal{S}^R \cap \mathcal{S}^B$ to the problem $\mathcal{S}_s^R \cap \mathcal{S}_s^B$.

both the surfaces $\mathcal{S}^R$ and $\mathcal{S}^B$ (Figure 11, left);

2. starting from this data structure, we extract the two sub-meshes $\mathcal{S}_s^B$ and $\mathcal{S}_s^R$: we first build the set $\mathcal{C}_s$ of the hexaedral cells where both the red and the blue triangles are stored together (Figure 11, middle); then we define $\mathcal{S}_s^B$ and $\mathcal{S}_s^R$ as the union of the triangles $T \in \mathcal{C}_s \cap \mathcal{S}^B$ and $K \in \mathcal{C}_s \cap \mathcal{S}^R$, respectively;

3. we apply the intersection algorithm in Section 2.2 to the reduced configuration $\mathcal{S}_s^R \cap \mathcal{S}_s^B$ to detect the intersection line Γ (see in Figure 11, right). Thus, the binary tree data structure is built for the sub-mesh $\mathcal{S}_s^R$ only.

Before considering realistic geological configurations, we assess the performances of this algorithm on the setting in Figure 12. The horizon $\mathcal{S}^R$ is characterized by a non uniform sizing, i.e., we distinguish a portion, the one on the left, where the mesh is rather coarse, in contrast to the part on the right where the triangles are rather refined. Formulas (3)-(4) lead to a hexahedral mesh with huge cells, containing a large number

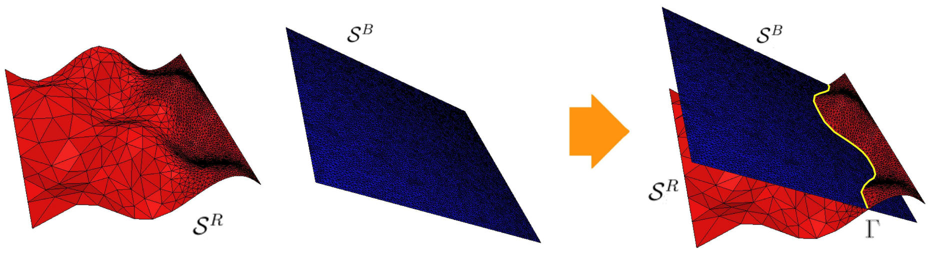

Figure 12: Intersecting non-coplanar horizons.

of elements where $\mathcal{S}^R$ is fine. Nevertheless, the reduction step yields the sub-meshes $\mathcal{S}_s^R$ and $\mathcal{S}_s^B$ consisting of 1415 and 2614 triangles in contrast with 4368 and 15519 elements for $\mathcal{S}^R$ and $\mathcal{S}^B$, respectively. As a consequence, the intersection $\mathcal{S}_s^R \cap \mathcal{S}_s^B$ based on the AB algorithm turns out to be quick: less than one second suffices to detect the intersection, to be compared with a timing of 3 and 74 seconds demanded by the AB and by the structured data search algorithm, respectively[2] when applied to the whole configuration.

Let us apply now the proposed algorithm to more realistic geological configurations: the presence of faults intersecting the geological horizons very often leads to really complex configurations (see Figures 13, left for two examples). In particular, a single fault may intersect an arbitrary number of horizons: the corresponding intersection lines assume any shape, depending on the displacement of the intersected horizons (compare the straight with the irregular yellow lines in Figure 13, right).
Figure 13 shows that the proposed algorithm works extremely well in practice, despite the possible highly complex configurations. Notice, for instance, the capability to detect the intersection between the green horizon and the brown fault in Figure 13, top-left, despite the fault slightly overcomes the horizon. The reduction step allows us to operate with a leaner data structure thus saving computational costs: about two seconds are demanded to identify the intersection for both the geological configurations. Due to the size of the problem, an approach based on a full data structure might be computationally prohibitive.

3 Detection of regions on a horizon

As shown in Figure 13, the complexity of the geological configurations we are interested in often leads to *composite* intersections, represented by the union of different piecewise linear curves (see Figure 14, left for an example). At this stage, the triangulated surface and the output of the intersection procedure are still separated entities, i.e., the intersection curve does not necessarily follow the edges of the triangulation

[2]Throughout the paper, all the computations have been carried out on a notebook with an Intel dual core CPU at 2.26 GHz with a 3 GB RAM.

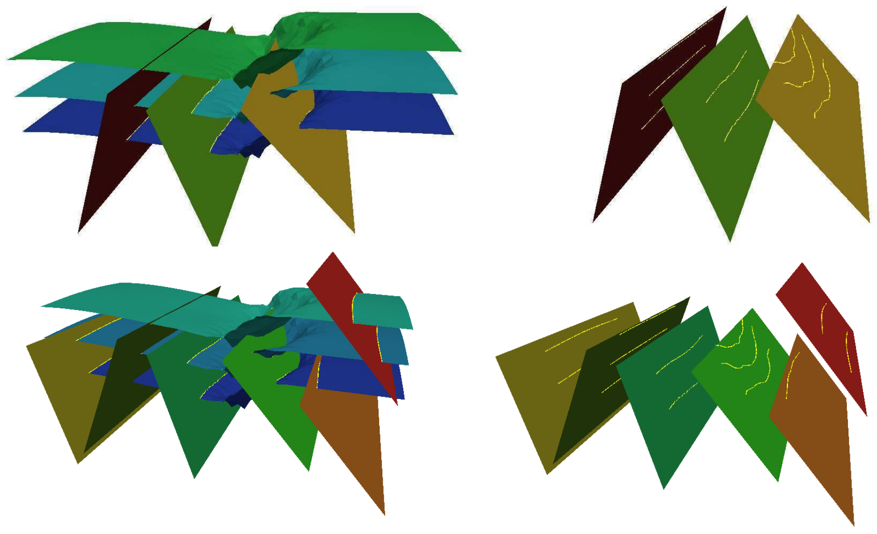

Figure 13: Possible geological configurations, where horizons and faults intersect (left); corresponding intersection lines identified on the fault surfaces (right).

but it may cross the mesh elements. Furthermore, the intersection curve identifies distinct regions on the horizon at hand (e.g., in Figure 14, right we recognize nine distinct areas).

Our actual goal is to detect such regions automatically. In particular, let $\mathcal{I}$ denote the intersection curve on a certain surface $\mathcal{S}$. We aim at finding a partition $\mathcal{P} = \{\omega_1, \omega_2, \dots \omega_n\}$ of $\mathcal{S}$ such that:

- $\bigcup_{i=1}^{n} \omega_i = \mathcal{S}$;
- $\mathring{\omega}_i \cap \mathring{\omega}_j = \emptyset \quad \forall i \neq j, \, i, j = 1, \dots, n$;
- $\partial \omega_i \subset \mathcal{I} \quad \forall i = 1, 2, \dots n$;
- $\mathring{\omega}_i \cap \mathcal{I} = \emptyset \quad \forall i = 1, 2, \dots n.$

The subdomains ω_i are assumed to be closed set and $\mathring{\omega}_i$ stands for the internal part of ω_i.

The approach we propose here consists of two distinct phases:

- we include the intersection curve $\mathcal{I}$ into the surface mesh via a suitable remeshing: the information of these two, a priori distinct, geometrical entities, is thus properly linked;
- we subdivide the triangulated surface into regions ω_i so that to define a partition $\mathcal{P}$ of $\mathcal{S}$ matching the properties above.

In the two next sections we deal with these two phases, separately.

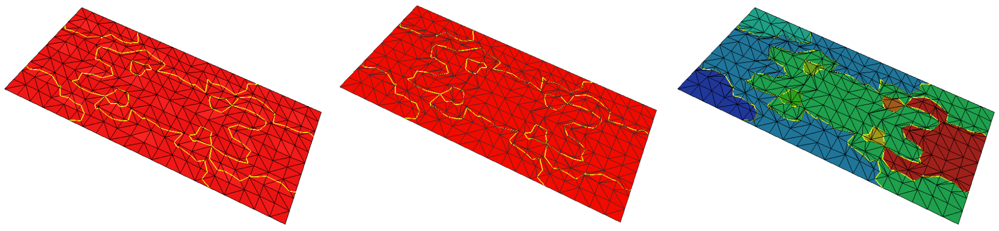

Figure 14: Example of composite intersection $\mathcal{I}$ (the yellow line on the left); inclusion of $\mathcal{I}$ into the mesh (middle); region detection (right).

3.1 Inclusion of the intersection curve

This step is quite complex since we cannot make any a-priori assumption on the shape of the intersection curve $\mathcal{I}$. We only know that the segments constituting the intersection line lie in some triangle of the surface.

The proposed method aims at properly remeshing each triangle crossed by $\mathcal{I}$ with the constraint of including the intersection segments in the new mesh. Fort this purpose, as first step, we have to find the elements crossed by $\mathcal{I}$: of course, no problem is yielded by the intersection segments coinciding with an edge of a triangle (or with a portion of it). Successively, we have to properly remesh each crossed element so that to guarantee the global conformity of the new mesh.

To detect the crossed triangles we can resort to the data structure search algorithms of Section 2: in particular, we employ the coupled approach of Section 2.3. To illus-

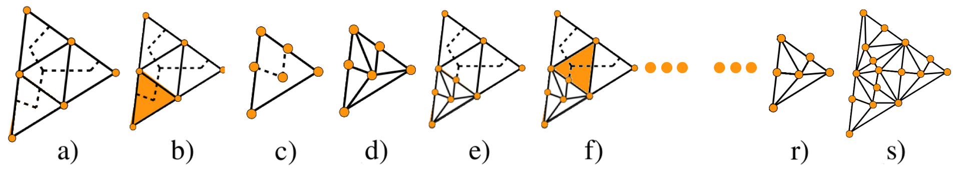

Figure 15: Example of inclusion of an intersection curve.

trate the procedure which creates a new triangulation conforming with the intersections, let us consider the four elements in Figure 15, left. We aim at adding the six dashed segments to the existing mesh to get a new mesh that includes them. As sketched in Figure 15, we process separately each of the four elements: the remeshing is performed first by including the intersection segments among the edges of the new mesh (Figure 15, c)) and then by adding additional edges to locally preserve the conformity of the mesh (Figure 15, d)). The conformity is inherited by the global mesh (Figure 15, s)). In Figure 14, middle we show the final result of the inclusion of the intersection curve procedure applied to the configuration on the left.

We remark that the quality of the elements thus generated is not necessarily good (really thin triangles might appear after the remeshing as shown in Figure 14, middle). We come back to this important issue in Section 4.

3.2 Subdivision of a horizon into regions

The inclusion of the intersection curve $\mathcal{I}$ into the mesh of the horizon at hand $\mathcal{S}$ makes the detection of the different regions a straightforward task. We assume again the simple configuration in Figure 15 as reference to explain the approach for the region detection.

We apply a sort of *diffusive* procedure. We assign a source to a certain triangle of the mesh; then we follow the diffusion of this source in the adjacent elements, driven by the triangle connectivity and by these simple rules:

1) each triangle spreads the source into the adjacent triangles across its edges;

2) the diffusion stops at the intersection segments.

In particular, we mean that two triangles are adjacent if they share an edge. Of course, at the end of the procedure, all the triangles of the mesh have to be assigned to a certain region ω_i. This means that we need to restart the procedure until we have partitioned the whole surface. We formalize this procedure via the following algorithm:

```
n = 0; i = 0; $\bigcup_{\mathtt{i}=1}^{\mathtt{n}} \omega_{\mathtt{i}}$ = $\emptyset$;
until $\bigcup_{\mathtt{i}=1}^{\mathtt{n}} \omega_{\mathtt{i}}$ == $\mathcal{S}$
        i = i + 1;
        choose $K \in \mathcal{S} \setminus \{\bigcup_{\mathtt{i}=1}^{\mathtt{n}} \omega_{\mathtt{i}}\}$;
        $\omega_{\mathtt{i}} = K$;
        until $\exists\ T : \partial T \cap \partial\omega_{\mathtt{i}} \neq \emptyset$ & $\partial T \cap (\partial\omega_{\mathtt{i}} \cap \mathcal{I}) = \emptyset$
                $\omega_{\mathtt{i}} = \omega_{\mathtt{i}} \cup T$;
        end
        n = n + 1;
end
```

Relation $\partial T \cap (\partial\omega_{\mathtt{i}} \cap \mathcal{I}) = \emptyset$ essentially checks if triangle T is on the correct part of $\mathcal{I}$. Figure 16 exemplifies the region subdivision algorithm on the intersected mesh of Figure 15, s). Three diffusive processess take place in such a case. In Figure 14, right we show the result associated with the horizon of interest: nine regions ω_i are now detected.

We highlight that the algorithm above as well as the approach proposed in in Section 3.1 can be extended, in a straighforward way, to more general frameworks, e.g., to quadrilateral meshes as well as to a higher dimension (for instance, to detect the volumes identified by a set of surfaces in a tetrahedral mesh).

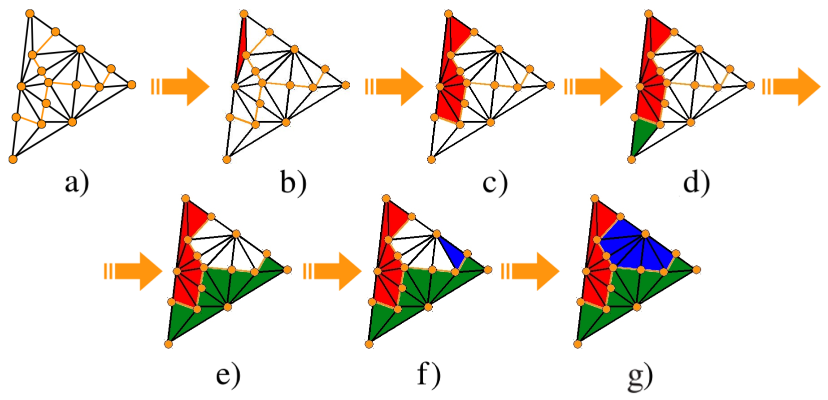

Figure 16: Example of region subdivision (the yellow curve marks $\mathcal{I}$).

4 Mesh quality improvement

The operations among surfaces in Sections 2 and 3 might lead to bad quality meshes with distorted elements. In particular, the inclusion of the intersection curve often yields really thin triangles. With a view to the approximation of a partial differential equation model (e.g., the Darcy equation) in a sedimentary basin of interest via a finite element scheme, the shape of the mesh elements may strongly affect the conditioning of the corresponding stiffness matrix. Meshes constituted by equilateral triangles usually guarantee reliable numerical results, while thin triangles often lead to badly conditioned matrices (see, e.g., [9]). For this reason, the geological reconstruction procedure proposed in this paper includes, as last step, suitable mesh modification modules to improve the shape of the mesh elements constituting the detected horizons and faults.

We resort essentially to the following four geometric operations (see Figure 17 for a corresponding schematic representation):

- *node smoothing*: we use the trapezium drawing (TD) approach proposed in [8], suited to deal with any not necessarily smooth surface;
- *edge swapping*: we choose the best diagonal for the quadrilateral formed by two adjiacent triangles, in order to minimize the maximum angle of such a quadrilateral; of course, the quadrilateral must be convex for edge swapping to be performed;
- *edge collapsing*: we remove vertices or edges (and also triangles via successive edge collapses) to avoid excessively short edges;
- *edge splitting*: a node is inserted at the midpoint of the longest edges to minimize the maximum angle of the mesh.

To quantify the shape of a generic triangle T, we resort to the so-called *quality index* $q(T) = 2\,r/R$ ([9, 10, 8, 4]), where r and R are the radius of the circumference

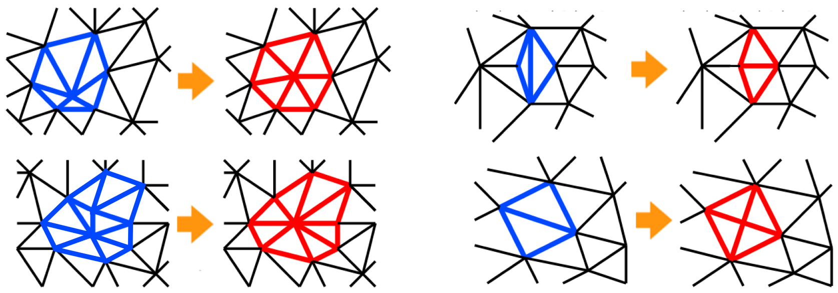

Figure 17: Geometric operations to improve the quality of the mesh: node smoothing, edge swapping, edge collapsing, edge splitting (top-bottom, left-right).

inscribed and circuscribed to T, respectively. If T is an equilateral triangle, $q(T) = 1$; viceversa, $q(T) \ll 1$ if T is a very stretched triangle.

Figure 18 shows the benefits led by the four mesh operations when applied to the surface in Figure 14, right. The number of the mesh elements is significantly increased (the original mesh has 1136 elements while the regularized mesh consists of 2726 triangles) but now all the triangles are very regular. The improvement of the mesh quality is confirmed by the hystograms in Figure 19, which represent the distribution of the quality index $q(T)$ on the mesh elements before and after the mesh regularization.

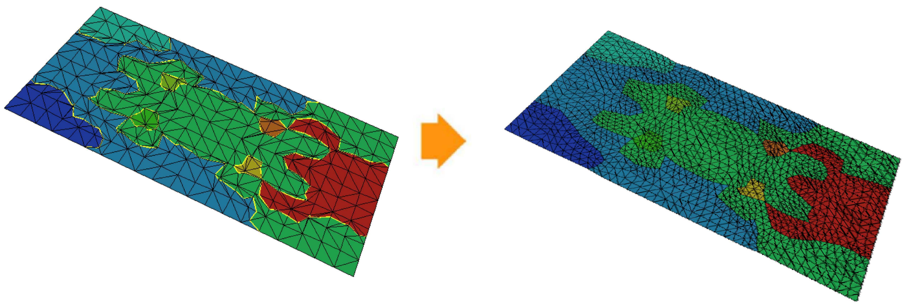

Figure 18: Instance of mesh quality improvement: original mesh (left) and improved mesh (right).

5 Generation of a geological geometry of interest

This section covers some other fundamental aspects that may be relevant when dealing with the geometrical modeling of geological basins. Namely, the requirement of completing possible missing data, the treatment of the so-called hard and soft horizons, and

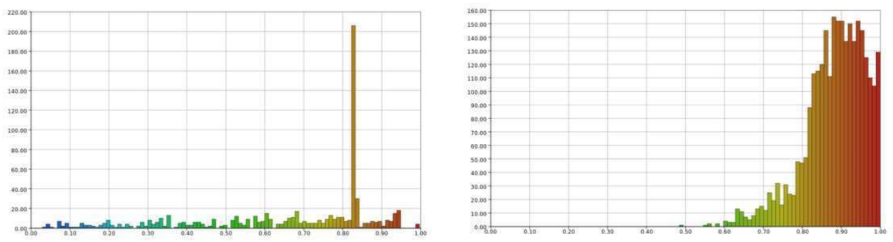
Figure 19: Distribution of $q(T)$ before (left) and after (right) the mesh quality improvement.

the construction of subvolumes.

5.1 Lack of data

The seismic data used for the simulations are often incomplete or non reliable due to either a lack of existing coverage or inadequate and old measurements. Thus, moving from the available information, it is sometimes necessary to reconstruct the horizons in the geological basin of interest before generating a corresponding tetrahedral mesh.
In particular, we investigate two different techniques to deal with a possible lack of data. As first approach, we have resorted to a well-known geo-statistical technique, i.e., the *kriging* ([2]). It is a regression method to recover the value of a certain field at unobserved locations starting from observations of the same field in nearby sites. In this case, the field of interest is the location of a horizon characterized by any lack of information (for instance, a hole). To apply this technique, we need to assume that the surface may be described explicitly as $z = f(x, y)$. We refer to Figure 20, left or Figure 22, left for possible examples.

In more detail, moving from a set $\{z_Q\}$ of z-coordinates associated with a set $\mathcal{N} = \{Q\}$ of points on the horizon with the lack of data, we recover the z-coordinate z_P of a point P located inside a hole as

$$z_p = \sum_{Q\in\mathcal{N}} \lambda_Q z_Q \,, \tag{9}$$

with λ_Q a suitable weight associated with the point $Q \in \mathcal{N}$. The type of kriging determines the choice for the unknown weights: we resort to a standard ordinary kriging where the weights essentially depend on the variogram associated with the starting data [2]. Moreover, the points in $\mathcal{N}$ are not necessarily spread on the whole horizon but they might be located only in a neighborhood of the point P. The computation of the weights λ_Q is not always an easy task: we resort to several searching processes that lead to the solution of several linear systems. Details on kriging may be found in the cited bibliography.

As an alternative approach we employ an implicit representation of the horizons based on radial basis functions ([3]): a horizon $\mathcal{S}$ is identified as the zero-level iso-surface of a suitable function $f : \mathbb{R}^3 \to \mathbb{R}$, i.e., $\mathcal{S} = \{P \in \mathbb{R}^3 : \ f(P) = 0\}$ with

$$f(P) = \sum_{Q \in \mathcal{N}} c_Q \phi(r), \tag{10}$$

where $r = ||P - Q||$ is the standard Euclidean distance between P and Q, ϕ is a radial basis function (in our simulations we choose $\phi(r) = r^3$), and c_Q are unknown coefficients determined by imposing interpolation constraints. This usually leads to solve an ill-conditioned full linear system of dimension $\mathrm{card}(\mathcal{N})$, which, therefore, has to be properly solved; yet we have eventually an implicit representation, which is more flexible than the one provided by the previous strategy.

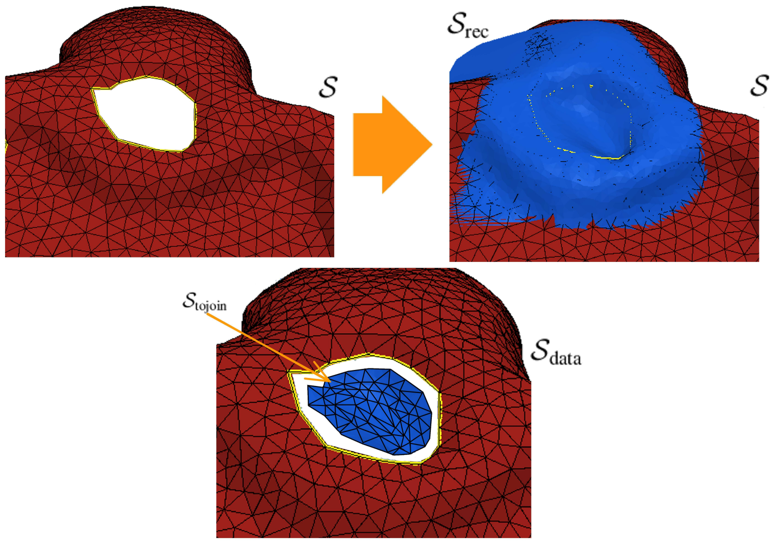
$\mathcal{S}$
$\mathcal{S}_{\text{rec}}$
$\mathcal{S}$
$\mathcal{S}_{\text{tojoin}}$
$\mathcal{S}_{\text{data}}$

Figure 20: Original horizon $\mathcal{S}$ with a hole (left); the recovered part $\mathcal{S}_{\text{rec}}$ overlapped to $\mathcal{S}$ (middle); the set $\mathcal{S}_{\text{tojoin}}$ (right).

Thanks to the numerical validation, we believe that the approach based on an implicit representation of the non complete horizon is more suited to deal with any kind of surfaces. In particular, this method allows us to treat in a more straightforward way also complex surfaces that cannot be expressed via a single-valued function f.

Now, independently of the technique adopted to recover the missing data, we aim at properly joining the recovered part $\mathcal{S}_{\text{rec}}$ of the horizon with the original horizon $\mathcal{S}$ (see Figure 20, middle). We remark that the boundary of the hole is approximated via a piecewise-linear function constituted by the edges of the elements of $\mathcal{S}$ around the hole itself. The goal is consequently twofold: first, we have to identify the elements of $\mathcal{S}_{\text{rec}}$ inside the hole; then, we have to add them to the triangles constituting $\mathcal{S}$. To do this, we proceed in such a way:

1. we build an auxiliary mesh $\mathcal{S}_{\text{bound}}$ around the hole and overlapped to $\mathcal{S}$: the idea is to consider, for each node of $\mathcal{S}$ along the hole, the direction normal to the surface and to fix two points along it, at a certain distance $\pm d$, respectively. These points along the normal directions allow us to build a strip constituted of couples of triangles, which represents $\mathcal{S}_{\text{bound}}$ (see Figure 21);

2. we find the intersection curve $\mathcal{S}_{\text{bound}} \cap \mathcal{S}_{\text{rec}}$ via the mixed structured-AB data search algorithm;

3. we add this intersection curve to $\mathcal{S}_{\text{rec}}$ following the approach in Section 3.1;

4. we identify the set $\mathcal{S}_{\text{tojoin}}$ of the elements of $\mathcal{S}_{\text{rec}}$ strictly cointained in the hole (see Figure 20, right) by employing a suitable variant of the *diffusive* procedure described in Section 3.2;

5. we join the meshes $\mathcal{S}_{\text{tojoin}}$ and $\mathcal{S}$, by properly connecting the outer rows of the triangles in $\mathcal{S}_{\text{tojoin}}$ and $\mathcal{S}$ so that to guarantee the conformity of the recovered mesh.

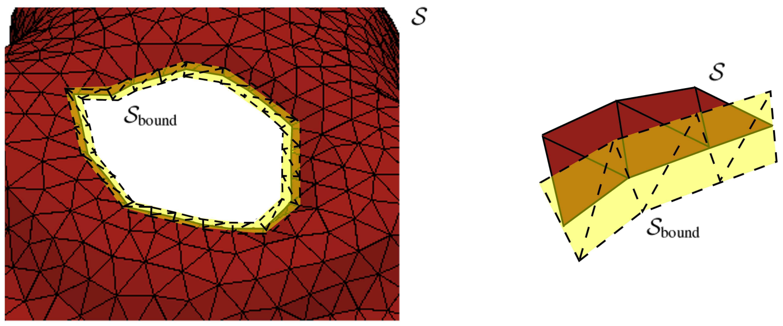

Figure 21: The auxiliary mesh $\mathcal{S}_{\text{bound}}$ associated with the hole in $\mathcal{S}$ (left); a zoom (right).

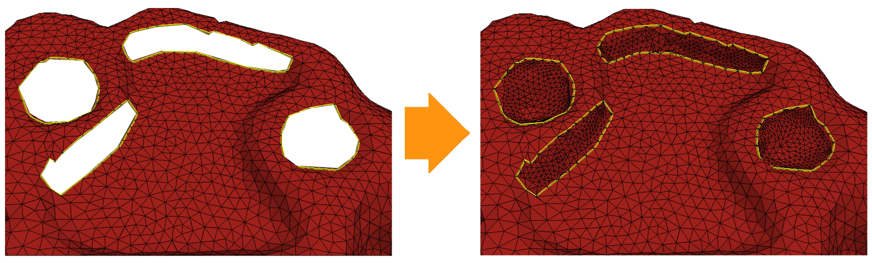

Figure 22: Reconstruction of four holes on a horizon of interest.

Figure 22 shows the result of this approach when several holes occur on a certain horizon. The approach based on an implicit representation of the horizons has been employed to build the surface $\mathcal{S}_{\text{rec}}$. As we shall see in the next sections, the management of geological horizons demands additional care in some circumstances.

5.2 Hard and soft rocks

Different kinds of rock do exist in nature. As a possible classification, we may distinguish them in hard and soft rocks. The hardness of a rock essentially depends on the nature of the grains constituting the rock as well as on what kind of natural glue holds them together: rocks baked in the deep underground are usually very hard (marble, for instance), while mudstones and shales are examples of soft rocks. The different nature of the geological layers overlapping in the basin of interest has to be properly taken into account when generating the corresponding geometry. If, for instance, a layer of marble stands above a layer of clay, we expect that the layer below is compressed by the layer above. Of course, direct measurements, such as core drillings, can be helpful in recovering these scenarios.

To deal with this possible interplay among horizons, we have set up an *ad hoc* procedure. Let us focus on the geological configuration in Figure 23, left: a hard horizon, $\mathcal{S}_h$, is compressing a soft horizon, $\mathcal{S}_s$. In the geometry of interest, a new surface, $\mathcal{S}_r$, replaces both the horizons $\mathcal{S}_h$ and $\mathcal{S}_s$. In more detail, we assume that $\mathcal{S}_r$ coincides with $\mathcal{S}_s$ in the regions where $\mathcal{S}_h$ lays over $\mathcal{S}_s$; viceversa, $\mathcal{S}_r$ coincides with $\mathcal{S}_h$ where $\mathcal{S}_h$ lays under $\mathcal{S}_s$ (see Figure 23, right). From an operative viewpoint, the computation of the intersection curve $\mathcal{S}_h \cap \mathcal{S}_s$ as well as the detection, on both $\mathcal{S}_h$ and $\mathcal{S}_s$, of the regions bounded by this curve are crucial steps for the generation of the horizon $\mathcal{S}_r$. The geometric operations in Sections 2–3 turn out to be consequently useful for this purpose.

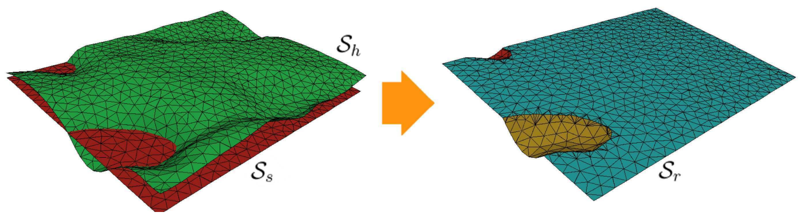

Figure 23: Starting geological configuration (left): a hard horizon (the green one) intersects a soft horizon (the red one); the horizon $\mathcal{S}_r$ which replaces $\mathcal{S}_h$ and $\mathcal{S}_s$ (right).

5.3 Selection of subvolumes

Very often we are interested in determining geological information related to a certain subsurface volume of interest rather than to the whole sedimentary basin (see Figure 24). This volume becomes the representative of the basin at hand. In particular, we aim at generating a tetrahedral mesh of this geological volume which takes into account the presence of any horizons and faults. Before proceeding with the volume mesh generation, suitable preprocessing procedures are sometimes demanded on the involved horizons and faults.

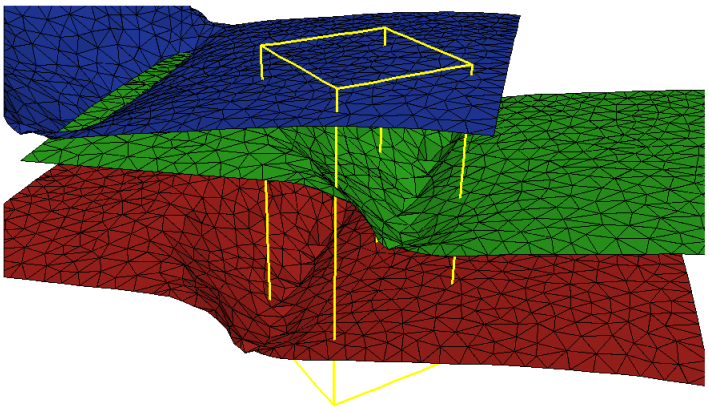

Figure 24: Example of volume of interest in a geological configuration.

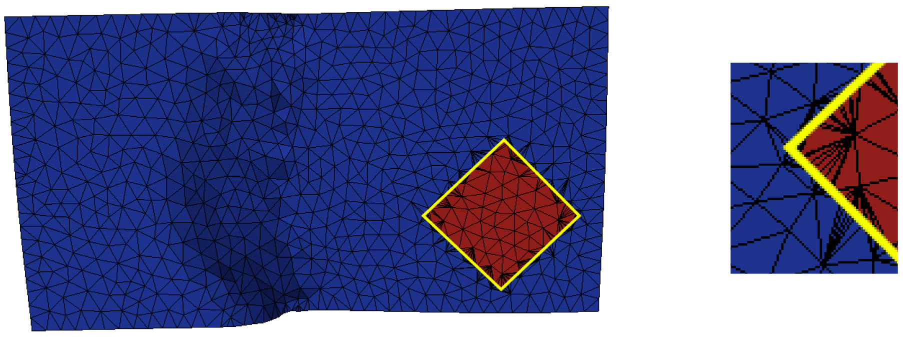

Figure 25: Portion (in red) of the blue horizon in Figure 24 cut by the volume of interest (left); zoom in on the corner leftmost (right).

Purpose of this section is to show how the geometric operations introduced in Sections 2–3 can be useful in this preprocessing phase. First of all, we are able, via the surface intersection and the region detection procedures, to identify, for each horizon, the corresponding portion cut by the volume of interest (see Figure 25, for an example). Starting from these cutouts on the different horizons, we can build the boundary of the geological volume.

Nevertheless, as highlighted in Section 4, the involved operations among surfaces often corrupt the quality of the mesh elements: this is evident in Figure 25, right as well as in Figure 26, right where a lot of stretched triangles appear. Thus, before dealing with the tetrahedral mesh generation, it is crucial to improve the mesh quality of horizons

and faults: we pursue this task essentially by exploiting the four geometric operations in Section 4. Finally, the 3D mesh is built using a generalized Delaunay procedure implemented in the TetGen library [7].
This whole procedure allows us to obtain a detailed representation of the geological volume of interest via a tetrahedral mesh of good quality which is constrained to the horizons and faults inside the volume. Figure 27 provides an instance of the outcome of the procedure when applied to a rather complex geological configuration. The regular shape of the tetrahedra is evident.

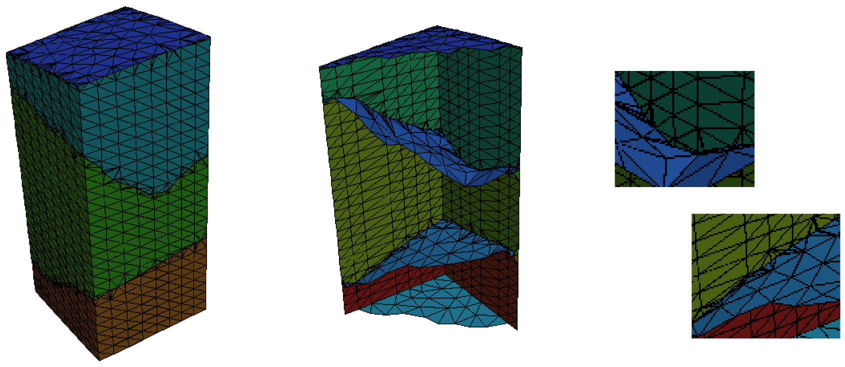
Figure 26: Example of a geological volume of interest (left); a corresponding vertical cutoff (middle); two zooms in on poor quality triangles (right).

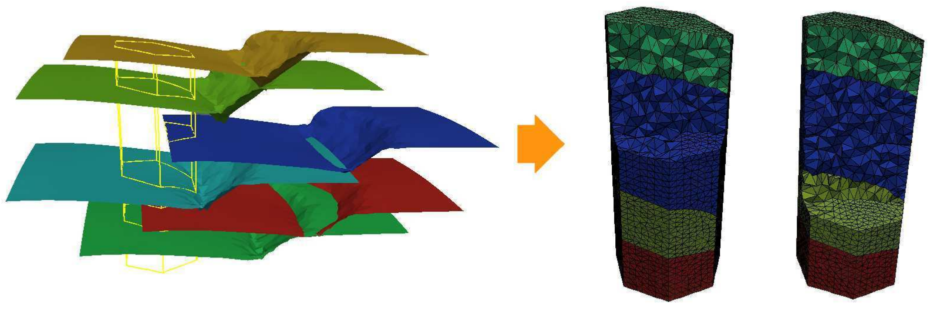
Figure 27: Example of tetrahedral mesh generation for the geological volume of interest on the left; two vertical cutoffs of the resulting volumetric mesh (on the right).

The procedure just described perfectly works when dealing with generic surfaces.

$\mathcal{T}$	STRUCTURED	AB-UNBALANCED	AB-BALANCED	MIXED	REGION DETECTION
$\simeq 6800$	2	3	2	2	1
$\simeq 68000$	8	121	45	13	2
$\simeq 680000$	30	35148	10114	67	19

Table 2: $\mathcal{S}_1^R \cap \mathcal{S}_1^B$: CPU time for the different surface intersection algorithms and for the region detection on uniform meshes.

6 Numerical assessment

This section is meant to provide a more quantitative analysis of both the surface intersection and the region detection procedures in Sections 2–3 on benchmark configurations.

We consider two pairs of intersecting surfaces with the aim of detecting the corresponding intersection curves $\mathcal{S}_i^R \cap \mathcal{S}_i^B$, for $i = 1, 2$, as well as the regions bounded by these (see Figure 28). The configuration $\mathcal{S}_2^R \cap \mathcal{S}_2^B$ can be assumed as the result of two sinusoidal surfaces which intersect each other. To approximate the four surfaces, we

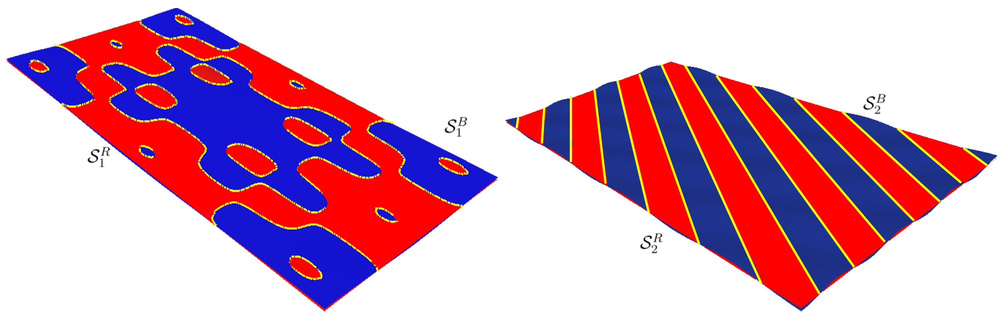

Figure 28: Intersecting surfaces: $\mathcal{S}_1^R \cap \mathcal{S}_1^B$ (left), $\mathcal{S}_2^R \cap \mathcal{S}_2^B$ (right).

resort to different families of meshes. We compare in terms of CPU time the performances of the surface intersection procedures addressed in Section 2. Finally, we apply the region detection strategy proposed in Section 3.
Tables 2 and 3 gather the results of such a comparison for uniform meshes of about 6800, 68000 and 680000 triangles, respectively. In particular, for the different surface triangulations, we collect the CPU time (in seconds) required by the intersection algorithm based on a structured data search (second column), by the intersection procedure exploiting a balanced (third column) and an unbalanced (fourth column) AB search tree, by the mixed structured-AB data search approach (fifth column) and, finally, by the region detection phase (sixth column).

The values in Tables 2–3 confirm, first of all, the importance of creating a balanced binary tree: by comparing the values in the third and in the fourth column, we recognize that the CPU time approximately triplicates in the case of an unbalanced binary tree

$\mathcal{T}$	STRUCTURED	AB-UNBALANCED	AB-BALANCED	MIXED	REGION DETECTION
$\simeq 6800$	2	5	3	2	1
$\simeq 68000$	20	142	49	50	3
$\simeq 680000$	62	22753	791	172	23

Table 3: $\mathcal{S}_2^R \cap \mathcal{S}_2^B$: CPU time for the different surface intersection algorithms and for the region detection on uniform meshes.

	STRUCTURED	AB-BALANCED	MIXED
$\mathcal{S}_1^R \cap \mathcal{S}_1^B$	593	27	14
$\mathcal{S}_2^R \cap \mathcal{S}_2^B$	624	30	18

Table 4: CPU time for the different surface intersection algorithms on nonuniform meshes.

(and it becomes almost 30 times greater for the second configuration approximated via the finest mesh).
Then, we remark that the approach proposed in Section 2.3 improves the performances of the intersection algorithm based on an AB tree, even though the tree is balanced: the gain becomes particularly evident for increasingly finer meshes and in the case of the first geometric configuration where the surface intersection is more localized. The selected sub-meshes $\mathcal{S}_{1,s}^B$ and $\mathcal{S}_{1,s}^R$ are small enough to speed up the AB tree search procedure. On the contrary, a more widespread surface intersection, as in the case $\mathcal{S}_2^R \cap \mathcal{S}_2^B$, does not necessarily lead to small sub-meshes, $\mathcal{S}_{2,s}^B$ and $\mathcal{S}_{2,s}^R$, with a consequent less significative reduction of the corresponding CPU times.
Moreover, both the Tables 2 and 3 suggest us that the best intersection algorithm is the one based on a structured data search for both the geometric configurations and for this kind of meshes. Finally, the region detection is a really cheap operation in terms of computational costs for both the configurations and for each of the meshes to be selected.

The conclusion drawn above about the better performances of the structured data search algorithm is no longer the same when considering nonuniform meshes (as already anticipated in Section 2.3 on a simpler configuration). The columns in Table 4 provide the CPU time demanded by the structured data search algorithm, by the balanced AB tree approach and by the procedure proposed in Section 2.3, respectively when dealing with nonuniform meshes. The surfaces $\mathcal{S}_i^B$ and $\mathcal{S}_i^R$, with $i = 1, 2$, are approximated via meshes consisting of about 20000 and 65000 elements, respectively. Figure 29 shows the corresponding surfaces of intersection where the nonuniform structure of the meshes is evident.

In the presence of nonuniform meshes, the mixed structured-AB data search approach turns out to be the most effective: a significative saving in terms of CPU time is guaranteed with respect to the balanced AB tree procedure and, as expected, it becomes

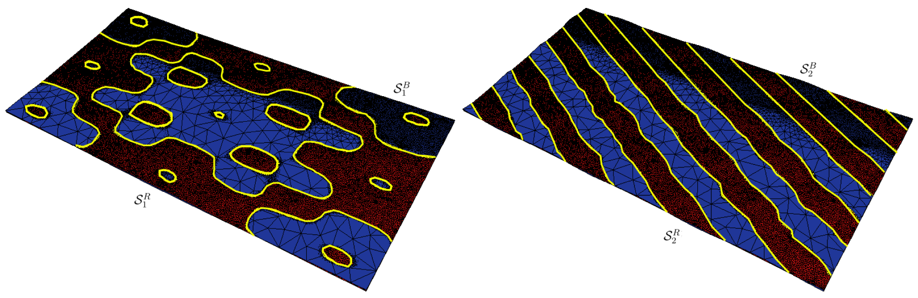

Figure 29: Intersecting surfaces and corresponding meshes: $\mathcal{S}_1^R \cap \mathcal{S}_1^B$ (left), $\mathcal{S}_2^R \cap \mathcal{S}_2^B$ (right).

even more remarkable with respect to the structured data search algorithm.

7 Conclusions

This paper illustrates various techniques applied to the geometric reconstruction of complex geological structures. Besides the specific target application, they are of rather general use.
We have demonstrated how by the combination of smart data structures and specific tools, it is possible to effectively treat complex geometries.
Moreover, we highlight that the proposed methods are rather flexible. Indeed we have illustrated these procedures on triangulated surfaces but many of them can be readily extended to different kinds of meshes.

We have implemented and compared different data structures, also in combination, to reach the conclusion that the best data structure strictly depends on the kind of involved meshes. If the surface meshes are structured or uniform, the structured data search turns out to be the most effective algorithm. Viceversa, the CPU times demanded by this straightforward approach can be large on nonuniform meshes. The mixed structured-AB data search approach proposed in Section 2.3 shows performances more homogeneous with respect to the type of the mesh. On structured and uniform meshes, the CPU times are comparable with the ones guaranteed by the structured data algorithm; on the contrary, we have shown that the computational gain led by the mixed procedure can become extremely relevant when dealing with nonuniform meshes. Therefore, the mixed approach is more suited for the general case.
The use of this new data structure has made it possible to analyze very complex geological situations at an affordable cost. In particular, we have combined algorithms specialized to identify the intersection of horizons and faults with a simple but effective algorithm to automatically detect the different regions forming the geological basin, completed with suitable mesh enhancing algorithms. This has allowed us to obtain good and conforming surface meshes for the different portions of the external and internal boundary of the basin of interest, ready to be input to a 3D mesh generator for

the production of meshes suitable for numerical simulations.

References

[1] F. Antonio, Faster Line Segment Intersection, Ch. IV 6, in Graphics Gems III (Ed. D. Kirk), Academic Press, San Diego 1992.

[2] N. Cressie, Statistics for Spatial Data, John Wiley & Sons Inc., New York, 1991

[3] A. Iske, Multiresolution Methods in Scattered Data Modelling, Lecture Notes in Computational Science and Engineering 37, Springer-Verlag, Berlin, 2004

[4] M. Longoni, Generation of high quality meshes on open surfaces for biomedical CFD, Degree Thesis 2007.

[5] T. Möller, A fast triangle-triangle intersection test, Journal of Graphics Tools 2 (2), 1997 25–30.

[6] H.Samet, Foundation of Multidimensional and Metric Data Structures, The Morgan Kaufmann Series in Computer Graphics and Geometry Modelling, Morgan Kaufmann Publishers Inc., Burlington MA, 2005.

[7] H. Si, A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator, Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany, 2006.

[8] J.B.Semenova, V.V. Savchenko, I.Hagiwara, Two techniques to improve the mesh quality and preserve surface characteristics, in Proceedings of the 13^{th} Meshing Roundtable, 2004 277–288.

[9] J. Shewchuk, What is a good linear element? Interpolation, conditioning and quality measures, in Proceedings of the 11^{th} Meshing Roundtable, 2002 115–126.

[10] J. Shewchuk, Robust adaptive floating-point geometric predicates, in Proceedings of the Twelfth Annual Symposium on Computational Geometry, 1996 141–150.