Unstructured Voronoi grids conforming to lower-dimensional objects

.Runar Lie Berge, Øystein Strengehagen Klemetsdal, Knut-Andreas Lie

Citation:
Berge, R.L., Klemetsdal, Ø.S. & Lie, KA. Unstructured Voronoi grids conforming to lower dimensional objects, Comput Geosci (2018). https:/doi.org/10.1007/s10596-018-9790-0

Noname manuscript No.
(will be inserted by the editor)

Unstructured Voronoi grids conforming to lower-dimensional objects

Runar Lie Berge · Øystein Strengehagen Klemetsdal · Knut-Andreas Lie

Received: date / Accepted: date

Abstract

We present a novel mixed-dimensional method for generating unstructured polyhedral grids that conform to prescribed geometric objects in arbitrary dimensions. Two types of conformity are introduced: (i) control-point alignment of cell centroids to accurately represent horizontal and multilateral wells or create volumetric representations of fracture networks, and (ii) boundary alignment of cell faces to accurately preserve lower-dimensional geological objects such as layers, fractures, faults, and/or pinchouts. The prescribed objects are in this case assumed to be lower-dimensional, and we create a grid hierarchy in which each lower-dimensional object is associated with a lower-dimensional grid. Further, the intersection of two objects is associated with a grid one dimension lower than the objects. Each grid is generated as a clipped Voronoi diagram, also called a perpendicular bisector (PEBI) grid, for a carefully chosen set of generating points. Moreover, each grid is generated in such a way that the cell faces of a higher-dimensional grid conform to the cells of all lower-dimensional grids. We also introduce a sufficient and necessary condition which makes it easy to check if the sites for a given perpendicular bisector grid will conform to the set of prescribed geometric objects.

R. L. Berge
University of Bergen, Norway
Tel.: +47 55 58 48 53
E-mail: runar.berge@uib.no

Ø. S. Klemetsdal
Norwegian University of Science and Technology, Norway
E-mail: oystein.klemetsdal@ntnu.no

K.-A. Lie,
SINTEF Digital, Norway
E-mail: Knut-Andreas.Lie@sintef.no

1 Introduction

The basic geometric description of a petroleum reservoir consists of a collection of surfaces representing stratigraphic layering and fault surfaces and horizons representing the structural architecture. These surfaces delineate the major compartments of the reservoir and often provide first-order control on in-place fluid volumes and fluid movement during production [3]. To correctly split the reservoir volume into sub-volumes, build flow units with similar or correlated petrophysical properties, and resolve flow patterns, it is important that a volumetric simulation grid conforms as closely as possible to these surfaces. Early simulation grids were either simple Cartesian boxes or block-centered grids used for dipping bedding, in which each rectangular cell could be compactly represented by four numbers (top depth and extent in each axial direction). These grid types are simple to construct, but cannot represent stratigraphy and structural architecture very well.

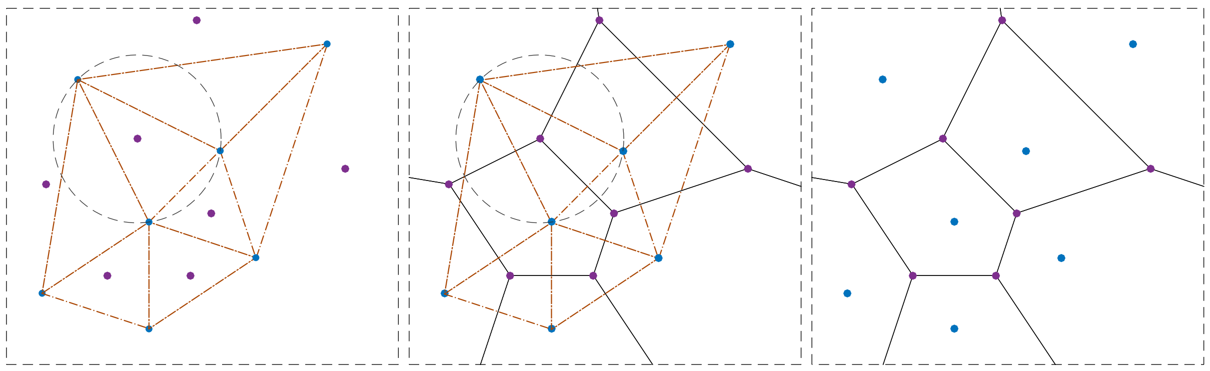
To better model sloping horizons, fault planes, and erosion surfaces, corner-point grids were introduced by Ponting [33]. These grids consist of hexahedral cells defined in terms of their eight *corner points*. The corner points are defined as pairwise depth values along four lines. Each line is defined by its end points, which are ordered lexicographically so that they form a quadrilateral areal mesh. These quadrilaterals will each have an associated vertical stack of cells forming a pillar that extends downward. In the simplest form, the coordinate lines are straight vertical lines distributed on a rectilinear areal mesh, giving rectangular pillars in which each hexahedral cell is delimited by six planar surfaces. More generally, the coordinate lines are sloping or curved lines defined over a curvilinear areal mesh, giving hexahedral cells delimited by bilinear planes. Depth values within each pillar are typically set so that the cell faces adapt to the stratigraphic layers of the reservoir. Each pair of depth values can collapse to a single point, so that cells can model erosion and even collapse to a surface of zero volume. The corner points of neighboring pillars are defined independently, and faults can thus be modelled by adapting the coordinate lines so that they follow major fault surfaces. The corner-point format has also been extended to include patches with local grid refinement to improve resolution e.g., in the near-well zone.

By construction, corner-point grids have an inherent Cartesian topology, which is advantageous for simulation. Unless the reservoir is heavily faulted or has extensive erosion, which both introduce non-neighboring connections, most cells will have six neighbors, which in turn leads to discretization matrices with reasonably regular sparsity patterns. On the other hand, corner-point grids are time-consuming to generate, require specialized software, have many subtle geometrical challenges caused by collapsing points and bilinear cell faces, and can easily give significant grid-orientation and grid-deviation effects [40] unless care is taken. The format is also inflexible and unable to accurately represent more complex features like y-shaped faults, thrust faults, and other overturned structures.

Wells have traditionally been described by the use of relatively simple semi-analytical models (inflow performance relationships) to capture the large pressure drop that takes place inside perforated grid blocks [31]. While such models may be sufficient for vertical or inclined well paths, they are often less suitable for directional wells, which may be highly deviated and consist of multiple branches and/or long and horizontal sections. Modern wells can also have (intelligent) inflow devices to control the fluid flow from the reservoir to the wellbore. Various techniques are also used to modify the near-well region to increase injectivity. Accurate and flexible description of well paths and increased grid resolution in the near-well region is crucial to evaluate and choose drilling, completion, and production strategies of advanced wells. Unfortunately, it is difficult to introduce new horizontal or deviated wells with suitable local grid adaption in a corner-point grid without changing the grid in large parts of the reservoir. Many have therefore started looking into more flexible unstructured grids to better capture complex fault or fracture systems and highly deviated well paths. Unstructured grids adapting to lower-dimensional objects are particularly important for so-called discrete fracture matrix (DFM) models of naturally fractured reservoirs [11, 18], in which the porous matrix is represented as a volumetric grid cell and the fractures are represented as lower-dimensional objects (surfaces or lines).

Unstructured grids were introduced in reservoir simulation in the late 1980s and early 1990s [9, 14, 15, 30]. The earliest techniques would embed refinements in a structured background grid in areas of interest. The perpendicular bisector (PEBI) grid is a popular choice for creating Voronoi type grids. The properties of PEBI-grids used for reservoir simulations are discussed by Verma and Aziz [39], whereas Courrioux et al. [5] were among the first to create a PEBI representation of a full-scale reservoir. The main drawback of these early gridding methods is their inability to represent complex structures, such as pinchouts and intersections of multiple faults. Later, Branets et al. [2] proposed a method that handles intersection of multiple faults and faults intersecting at sharp angles. A similar method is also presented by Manzoor et al. [24], Toor et al. [38]. Both methods create a protection layer around faults by use of constrained Delaunay triangulation and are thereby able to recover the faults exactly. Pinchouts and intersecting faults/fractures planes are treated by mirroring PEBI-sites (seed points for cell centers used in the grid generation) around the tracked features. A disadvantage with these methods is that they tend to give to congested PEBI-sites around these features.

In an attempt to countermand these problems, Ding and Fung [6] introduced a conflict-point removal scheme. The method starts by creating a structured background grid and placing a set of PEBI-sites equidistant around each fault/fracture plane to be tracked. Each PEBI-site is given a priority, and when two sites are too close, the site with lowest priority is removed. The generated grid conforms to the tracked planes and has fairly uniform cells. Recently, we extended the method to also generate conforming cells at intersections [20], thereby giving a robust method for generating Voronoi grids with control-point alignment of cell centroids and boundary alignment of cell faces in 2D. In 3D, the method only guarantees full alignment away from intersections. A different approach is taken by Merland et al. [26, 27], who suggest to place the PEBI-sites by an optimization method that minimizes the volume of cells cut in two by a fracture. This method is promising, even though one often needs to treat the grid manually after the optimization. Especially cells at fracture intersections can have undesirable geometries, and fracture planes are not reproduced exactly like in [20]. Another optimization method was proposed by Sun and Schechter [36] for discrete fracture-network models to reduce highly skewed cells and ensure good grid quality around fracture tips and intersections and in regions of high fracture density. See also Filippov et al. [8] for a dynamic gridding method in which the Voronoi grid is rearranged locally to account for opening/shutting of wells and fracture growth.



(a) Delaunay triangulation. (b) Delaunay triangulation and Voronoi diagram (c) Voronoi Diagram.

Fig. 1: Construction of a Voronoi grid from a point set by use of Delaunay triangulation. We start by specifying a set of generating points (blue dots), and then compute the corresponding Delaunay triangulation, defined so that no generating point falls inside the circumcircle of the triangles in the triangulation. The circumcenter (purple dots) of each triangle lies at the point where the three perpendicular bisectors of the triangle intersect. The Voronoi diagram is obtained by connecting the circumcenters along the perpendicular bisectors; hence the name PEBI. By construction, the Voronoi diagram extends to infinity and must therefore be clipped against the domain boundary to define a finite grid (here shown as dashed lines).

Many authors have also studied triangular grids adapting to faults and fractures. A complete review is out of our current scope, but we mention Brewer et al. [4], who present a method for exactly representing fractures by a triangulation. Methods for approximating faults and fractures by triangles have also been investigated [16, 19, 28]. Another method that has gained popularity in the latest years is the cut-cell method [12, 13, 23]. This method generates a grid by creating a mapping from a Cartesian grid to the physical domain, and then creates general polyhedral cells by cutting the Cartesian cells by crossing fractures.

In this paper, we present a novel method for generating unstructured Voronoi-grids in arbitrary dimensions conforming to fractures, faults, well paths and other types of lower-dimensional objects. All objects are assumed to be piecewise affine, and for some of the algorithms affine. We will look at two different conformity requirements: (i) structures that should be traced by faces of the grid, and (ii) structures that should be traced by cell centroids. Typically, wells or volumetric representations of fractures should be traced by cell centroids, whereas lower-dimensional objects such as fractures (in discrete-fracture matrix models), faults, horizons, and erosion surfaces as well as other types of internal boundaries should be traced by cell faces. The special feature of our method is that the grid is built dimension-by-dimension, starting with endpoints and intersections between constraining lines (and surfaces) in 0D, which deliminate 1D discretizations along line constraints, which in turn deliminate 2D discretizations of constraining surfaces. By building the grid this way, we ensure that it preserves intersections between lower-dimensional constraints and that the cells from a $(d-1)$-dimensional constraint will be faces in the d-dimensional grid. Our new method and its predecessors described in [20] have been implemented as a separate module called `upr` in the Matlab Reservoir Simulation Toolbox (MRST), which is an open-source community code designed for rapid prototyping and validation of new models and computational methods for simulating flow in porous media [21].

2 PEBI-grids

One popular approach to generate Voronoi grids is to construct them by first creating a Delaunay triangulation of a set of generating points and then construct the Voronoi grid by intersecting the perpendicular bisector lines/planes of the Delaunay triangulation, clipped against the domain boundary. Figure 1 illustrates this procedure. By careful placement of the points generating the Delaunay triangulation, we can control the perpendicular bisectors and hence how the Voronoi grid adapts to lower-dimensional constraints. In the petroleum literature, such grids are usually referred to as perpendicular bisector (PEBI) grids, whereas the term Voronoi is most common in other fields of science. In the following, we will use the two terms interchangeably.

This section explains the process of generating PEBI-grids conforming to control points and to internal boundaries. To simplify the discussion, we introduce some more terminology. Let $S_d = \{\mathbf{s}_i\}_{i=1\ldots n}$ be a set of generating points (or *sites* for brevity) in $\mathbb{R}^d$. The subscript on S_d is dropped if it is obvious from the context which dimension we are referring to. In all algorithms, we work with three different sets of sites: well sites, fracture sites, and reservoir sites. Well and fracture sites are created to make the grid conform to control points and internal boundaries, respectively. Reservoir sites are all other sites that generate the background grid.

To define the PEBI/Voronoi grid, we say that a point $\mathbf{x}$ belongs to a Voronoi cell Ω_i if it is at least as close to $\mathbf{s}_i$ as any other sites in S; that is, $\Omega_i = \{\mathbf{x} : \mathbf{x} \in \mathbb{R}^d, |\mathbf{x} - \mathbf{s}_i| \leq |\mathbf{x} - \mathbf{s}_j|, j = 1, \ldots, n\}$. The PEBI-grid is then defined as the set of all Voronoi cells. This grid is the dual of the Delaunay triangulation of the set of well, fracture, and reservoir sites, as proven in Appendix B. In the following, we only consider so-called clipped Voronoi diagrams, for which the infinite Voronoi diagram is restricted to a bounded set $\Omega \subset \mathbb{R}^d$ by clipping the outmost cells so that $\Omega_i^{\text{clipped}} = \Omega_i \cap \Omega$; see Berge [1] or Yan et al. [41] for more details. Figure 1 shows the most important duality properties between PEBI-grids and Delaunay triangulations. A thorough description of PEBI-grids and Delaunay triangulations is out of the scope of this paper. Instead, we refer to the textbook by Shewchuk et al. [35].

In the rest of the section, we first introduce our control-point conformity, which ensures that certain cell centers in the grid conform to curvilinear paths. The primary example would be a deviated well path. Next, we present our novel approach to generate grids where the faces conforms to $(d-1)$ dimensional objects, as well as to the $(d-i)$ dimensional intersection of these objects for $i \leq d$. The primary examples here would be intersections of fault surfaces or fractures. For brevity, the two types of conformity are henceforth referred to as wells and fractures for simplicity, even though the same techniques can be used to adapt to other lower-dimensional objects like faults or horizons. Once the two types of conformity have been introduced, we step back and present an improved algorithm that works for 2D grid and 3D grids with triangulated face constraints, before we discuss how to optimize this algorithm to improve the cells around fracture–fracture, fracture–well, and well–well intersections. Finally, we discuss various methods for generating reservoir sites.

2.1 Well sites.

In reservoir simulation, the diameter of the wellbore will typically be much smaller than the size of the grid cell. To account for the significant pressure variation from

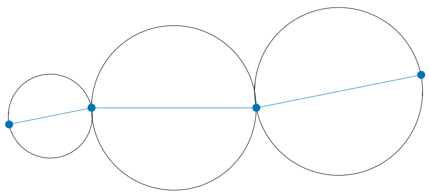


Fig. 2: The well condition is satisfied if the interior of each circle does not contain any sites.

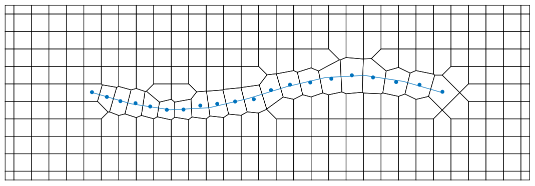


Fig. 3: A grid adapting to the path of a single well. The blue line is the well path, and the blue points the well cell centroids.

the sandface to the wellbore, which for most field and sector models takes place on a subscale inside each perforated cell, it is common to use an analytical or semi-analytical inflow performance relationship of Peaceman type [31]. These models are most accurate when the wellbore passes through the centroid of each perforated cell. To maximize accuracy, a good grid should therefore trace wells using cell centroids. In a high quality PEBI-grid, the cell-centroids coincide with their respective sites [7]. We therefore place a set of well sites along each well trajectory with the distance between consecutive sites given by a user-defined function. Because PEBI-grids are created as the dual of a Delaunay triangulation, we therefore require that consecutive well sites belonging to the same well segment should be connected by edges in the Delaunay triangulations of the sites. Consecutive well sites will then be neighbors in the dual PEBI-grid.

Definition 1 (Well condition) *If* $\mathbf{s}_1$ *and* $\mathbf{s}_2$ *are two consecutive well sites, the* **well condition** *is satisfied if the circle centered at the midpoint of the two sites and intersecting both of them does not contain any other sites from* S.

Circles defining the well condition are shown in Figure 2. When the well condition is satisfied, the line segment between $\mathbf{s}_1$ and $\mathbf{s}_2$ is so called *strong Delaunay* and will be an edge in the associated Delaunay triangulation [35]. Further, from the duality of PEBI-grids and Delaunay triangulations, the neighbor edge in the PEBI-grid will contain the midpoint of the edge [1]. Figure 3 shows a grid adapted to a single curved well trajectory, in which the distance between well sites in-

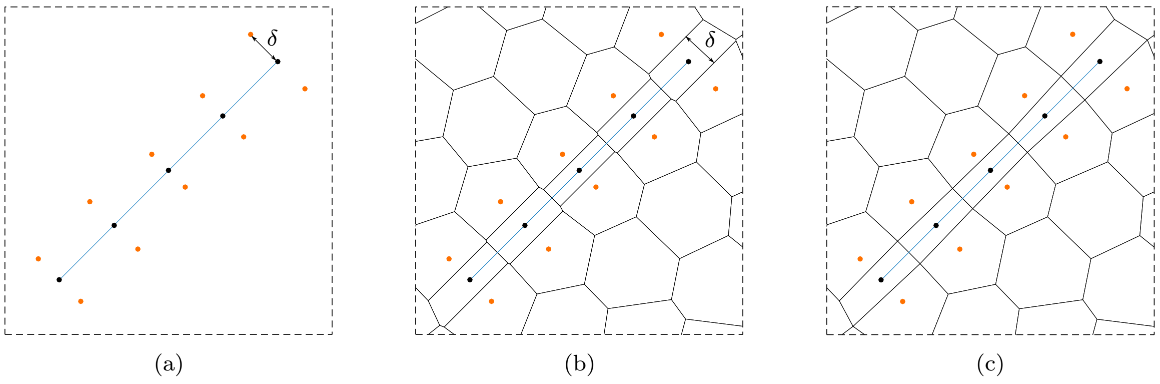


Fig. 4: Illustration of a trajectory with a protection layer. Black points are well sites, while orange points are protection sites. The distance δ from a well site and its protection sites also equals the diameter of the corresponding well cell. In (c) we have postprocessed the grid and removed any short edges. Unlike the unprocessed grid in (b), the processed grid in (c) is not strictly PEBI locally around the well.

creases slightly along the well path. Because the well condition is satisfied, neighbor edges between two consecutive well sites always intersect the well trajectory.

To better capture the symmetric flow in/out of wells, Fung et al. [10] suggested to add a protection layer around the well trajectories to make well cells with more regular shapes. Sun and Schechter [36] showed that one can create cells that explicitly represent the radius of a well by adding protection sites around the well sites. To add a layer of protection sites in 2D, we trace each well trajectory and place the protection sites normal to the resulting curve. Each well site will have two protection sites, one on each side. Figure 4 shows one well with a protection layer. The distance δ the protection sites are placed from the curve also equals the diameter of the corresponding well cell. We allow the distance δ to vary along the curve. This is also practical if we for example wish to create explicit volumetric representation of fractures with varying width [36]. In the figure, the distance function is perturbed slightly for each set of protection sites for illustration purposes. As noted by Klemetsdal et al. [20], this may introduce very short faces and these should preferably be eliminated as shown in Figure 4c, to make the grid more suitable for discretization of flow equations.

2.2 Fracture sites.

The final goal in this section is to obtain a PEBI-grid in which the faces of the cells conform to fractures. However, before we start describing the steps in our gridding algorithm, it is useful to understand its motivation. The faces conforming to a fracture will define a lower-dimensional grid of the restriction of the domain to the fracture. For a reservoir in 3D, the conforming faces will define a 2D grid of the fracture. Equivalently, for a 2D reservoir, the conforming faces will define a 1D grid. In this way, we can create a hierarchy of grids. If the reservoir is of dimension d, each fracture can be described by a grid of dimension $d-1$, whereas the intersection of two fractures can be associated with a grid of dimension $d-2$. In general, the non-degenerate intersection of n fractures can be associated with a grid of dimension $d-n$. This structure gives us a natural way of building the d-dimensional grid from bottom up: First, the zero-dimensional grids are built. From this, we build one-dimensional grids, then two-dimensional grids, and finally the three-dimensional grids. We note that the task of creating a conforming d-dimensional grid is reduced to the following problem: Given a grid G_{d-1} of dimension $d-1$ defined by the sites S_{d-1}, how can we create a PEBI-grid G_d of dimension d so that the faces of G_d conform to the cells of G_{d-1}?

To answer this question, we will first give a necessary and sufficient condition on the sites S_d of G_d. We will later see that an algorithm for generating conforming grids in 2D follows naturally from this condition. Repeating the same construction in 3D is not that straight forward, but the condition will give us a way to prove that a simpler algorithm works. We can also easily check whether our grid will conform to a fracture or not, even before we have created the grid. Let f_p be a facet of dimension p from the grid G_{d-1}. This means that f_p is a cell of G_{d-1} for $p = d-1$, a face (or edge) for $p = d-2$, and a vertex for $p = 0$. See Figure 5 for an illustration. The facet f_p is also a facet in G_d if there exists a set of at least $d-p+1$ sites from S_d such that for any point on f_p, we can draw a ball centered at this point, intersecting all sites of the set, and the interior of the ball does not contain any sites

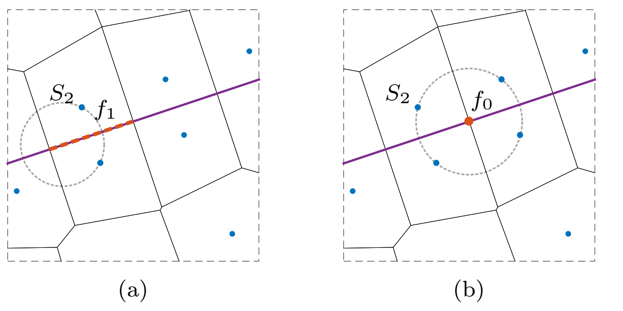


Fig. 5: The p-dimensional facet f_p of grid G_{d-1} exists in grid G_d if for at least $d-p+1$ sites in S_d, any ball centered in f_p, intersecting the sites, does not contain any sites from S_d. In (a), f_1 is a cell of the purple 1D grid. Any circle centered in f_1, intersecting the neighbour sites, does not contain any other sites in S_2 (blue dots), thus f_1 is a face in the 2D grid. In (b), f_1 is a vertex of the purple 1D grid. The circle intersect four sites from S_2, but contains none in its interior, thus f_0 is also a vertex in the 2D grid.

from S_d. This condition can easily be deduced from the definition of a PEBI-grid. Further, if G_d conforms to G_{d-1}, this condition must be satisfied for all facets in G_{d-1}. To check this condition for all facets is quite cumbersome, especially for the higher dimensional facets. Luckily, under two assumptions, we can simplify the condition and only consider the vertices of G_{d-1} (i.e., f_0). For each cell in G_{d-1}, we associate the two sites in S_d that have f_{d-1} as the face between them. The first assumption we make is that the union of the balls centered at the vertices of f_{d-1} and intersecting these two sites contains any other ball intersecting the sites and centered anywhere in f_{d-1}. The second assumption is that a vertex in G_{d-1} should be connected to at least d cells in G_{d-1}. Note that both of these assumptions hold true if G_{d-1} is a PEBI-grid, but is also valid for other grids, e.g., triangle grids. It is then necessary and sufficient to only check these balls around each vertex:

Definition 2 (Fracture condition) *Let* $\mathbf{s}_1$ *and* $\mathbf{s}_2$ *be two sites in* S_d *associated with a cell* c_{d-1} *from* G_{d-1}*. For each of the vertices of* c_{d-1}*, we draw a ball centered at it which intersects* $\mathbf{s}_1$ *and* $\mathbf{s}_2$*. The* **fracture condition** *is satisfied if the interior of all of these balls does not contain any site from* S_d*.*

If the fracture condition is satisfied for all cells in G_{d-1}, the grid G_d is guaranteed to conform to the lower dimensional grid. It is worth noticing that there should be one unique ball around each vertex. This means that if v is a vertex and $\mathcal{C}^v_{d-1}$ is the set of all cells from S_{d-1} which are connected to v, the fracture condition must be satisfied with the same radius for all cells in $\mathcal{C}^v_{d-1}$.

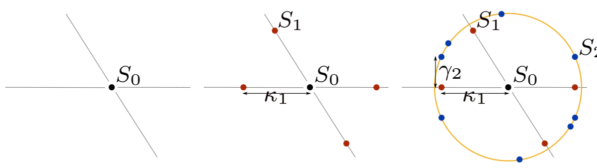


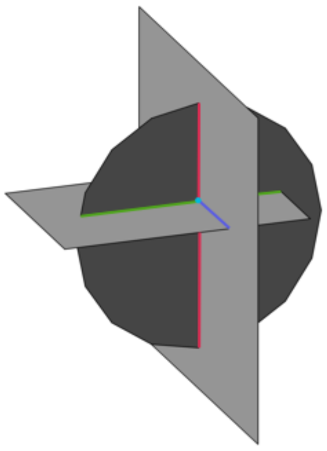
Fig. 6: Sites around an intersection. Black point is the 0D site, red points the 1D sites, and blue points the 2D sites.

At the boundary of G_{d-1} the second assumption above is not necessarily valid for a PEBI-grid anymore. A vertex at the boundary may be connected to d or fewer cells, e.g., see the vertices on the dashed boundary of Figure 5 which are connected to one or two cells. We therefore need to add more sites around each vertex on the lower-dimensional boundary. There are two requirements on the position of these vertices: They should lie on the ball from the fracture condition, and they should not violate the fracture condition for any other balls. We will discuss this in more detail in the 2D gridding below.

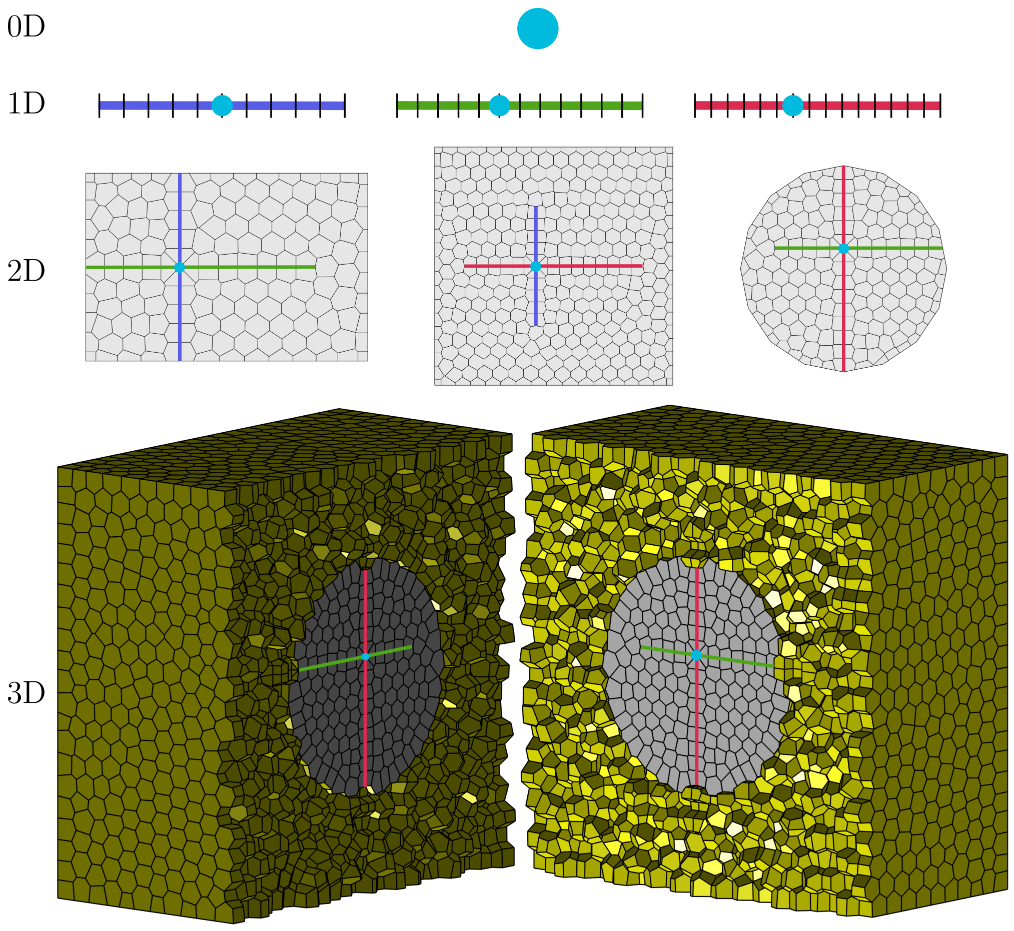
2.3 Algorithm for fracture sites in arbitrary dimensions.

As mentioned above, we build the grids in order of dimension, from the 0-dimensional grids through the d-dimensional grid. In the above discussion we did not require that the lower dimensional grid G_{d-1} to be PEBI, and below we will discuss the special case when it is a triangular grid. However, in this section we will present an algorithm where the grids in each dimension will be PEBI and built in such a way that the faces of the grid of dimension d correspond to the cells of the grid of dimension $d-1$. For $d=3$, this means that the cells of the 0D grids will be faces of 1D grids, the cells of the 1D grids will be faces of the 2D grids, and the cells of the 2D grid will be faces of the 3D grid. Equivalently, 0D, 1D, and 2D cells will be vertices, edges, and faces, respectively, in the 3D-grid. In this section we will assume the fractures are planar.

Assume we are given the sites S_{d-1} of a lower-dimensional PEBI-grid G_{d-1}. The steps in the following algorithm are demonstrated in Figure 6. For each site in S_{d-1}, we make two duplicates and move them a step-length γ_d in the positive and negative normal direction of the corresponding cell. Using the fracture condition above, it is easy to show that the cells of G_{d-1} will now be faces in G_d: Since G_{d-1} is a PEBI-grid, the distance from a vertex to the sites associated to it, will have the same distance, κ_{d-1}. This can be



(a)



(b)

Fig. 7: Intersections of three surfaces in 3D are shown in (a). (b) shows, in descending order, the corresponding 0D, 1D, 2D and 3D grids. The 3D grid is opened along the disk-surface.

seen in Figure 1b where the dashed circle has radius κ_2. The distance from all associated sites in S_d to the vertex will then be $\kappa_d = \sqrt{\kappa_{d-1}^2 + \gamma_d^2}$. The fracture condition then says that as long as all the interior of these balls around the vertices of G_{d-1} are empty, all the facets of G_{d-1} will be contained in G_d. To create a conforming grid we see that there are two things we have to be careful about. We are given some restrictions on where we can place the reservoir and well sites as to not violate the fracture condition. Also, we need to choose γ_d small enough so that the sites from one fracture does not interfere with the fracture condition on other fractures. It is also worth noting that κ_{d-1} will in general be different for each vertex, but γ_d should be chosen the same for all grids of dimension d connected by an intersection. If it was chosen to be different for two grids that share a vertex, the fracture condition would be violated for this vertex. We will see later that we easily can relax this requirement for the special case of 2D and certain cases in 3D, but in general, changing radius of one ball would require us to solve a non-linear problem to find the radii of all other balls.

In the implementation, we start with the 0-dimensional grids G_0, which represent intersections of at least d fractures. S_0 is just a point and trivial to grid. By using the above approach we can grid the 1D grids; for each of the 0D grids, we find the corresponding 1D lines that created this 0D point. We place a set of sites a distance $\pm\gamma_1$ from the intersection point, while the rest can be placed however we like, as long as the fracture and well condition is not violated. To generate the S_2 sites we take the S_1 sites and place them a distance $\pm\gamma_2$ from the 1D lines. Last, we take the S_2 sites and place two duplicates on each side of the fractures with a distance γ_3. Notice that we never used information about the vertices, edges, or faces of the lower-dimensional grids. The only information we need is the location of the S_{d-1} sites and the normal vector of the plane. We can therefore avoid creating any of the lower-dimensional PEBI-grids to save some computational time. The cost of this is that we are not able to check if the spheres around each vertex have empty interior. A simple case of three intersecting fractures is shown in Figure 7.

2.4 Algorithm for fracture sites conforming to simplices.

If we are in the special case that all cells of the lower-dimensional grid G_{d-1} have exactly d vertices (i.e., they are simplices), we are in luck as we now much easier can manipulate the radii of the fracture condition spheres.

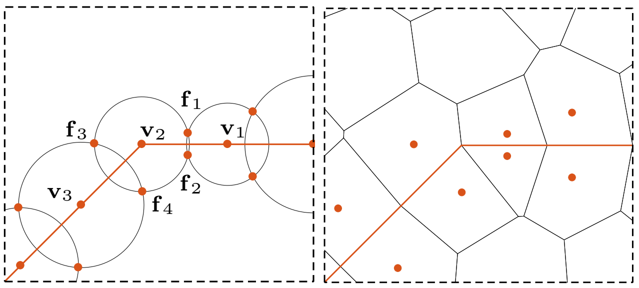


Fig. 8: The creation of fracture sites $\mathbf{f}_1$, $\mathbf{f}_2$, $\mathbf{f}_3$, and $\mathbf{f}_4$, from the 1D vertices $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$. The fracture condition is satisfied if the interior of the three circles does not contain any sites.

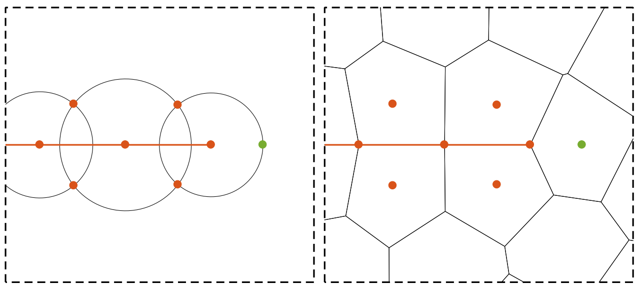


Fig. 9: Treatment of boundary vertices. For the boundary vertex of the 1D grid we need to add an extra tip site shown as a green point. This should be placed on the circle of the boundary vertex.

This is especially relevant for 2D gridding, as a 1D cell has exactly two vertices. It can also be relevant for 3D gridding if the fractures are given by a triangulation. For this section we assume that the fractures are piecewise planar over each simplex.

Let us first restrict ourselves to the 2D case ($d = 2$), and see how the fracture condition gives us a straightforward algorithm to find the sites S_2. In this case, a fracture is described by a line. The line is divided up in line segments, which defines 1D cells. We let $V = \{\mathbf{v}_i\}$ be the vertices of the 1D grid ordered such that cell c_i has vertices $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$. We draw a circle around each vertex, and require the two circles of a 1D cell to intersect, which gives us an upper and lower bound on the radii of the circles:

$$|R(\mathbf{v}_i) - R(\mathbf{v}_{i+1})| \leq d_i \leq R(\mathbf{v}_i) + R(\mathbf{v}_{i+1}). \qquad (1)$$

Here, d_i is the distance between the two vertices $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$. The fracture sites $\{\mathbf{f}_j\}$ are placed where two circles intersect. If R_i is small, the fracture sites will be placed close to the fracture curve, and if R_i is large, the fracture sites will be placed far from the fracture curve. In Figure 8 we see a single fracture where the radii has been chosen randomly for each vertex. By construction, the site pairs $\mathbf{f}_1\mathbf{f}_2$ and $\mathbf{f}_3\mathbf{f}_4$ satisfy the fracture condition as long as we do not place any well or reservoir sites inside the generating circles.

For the boundary vertices of the 1D grid this procedure only adds two sites, but to define a vertex in a 2D grid we need at least three sites. Thus, a third site has to be added for each boundary vertex. As mentioned in the section about the fracture condition there are two requirements for the position of the site; it should be placed on the circle drawn around the vertex, and it should not lie inside any of the other circles. We have chosen to place the site on the intersection of the circle and the tangential line of the 1D grid as shown in Figure 9.

We will discuss 2D gridding in more detail below, in particular how the flexibility of being able to freely choose the radii enables us to obtain high-quality cells around fracture intersections. However, first we note that we can generalize this algorithm for fractures defined by simplices in higher dimensions. In this paper we do not discuss how to generate a simplex grid of a $d-1$ object, but assume it is given. We start by drawing a sphere around each vertex of the simplex grid and place the sites at the intersection of these spheres. Since each lower-dimensional cell has exactly d vertices, the intersection of the corresponding d spheres gives us exactly two points. If we place the fracture sites of S_d at these intersections, the fracture condition is satisfied. We can then freely choose the radii of the spheres, but of course bounded by the requirement that the spheres should intersect. It is crucial that the lower-dimensional grid only consist of simplices; if a cell has $d+1$ or more vertices we can not choose the radii of each sphere independently, as the intersection of $d+1$ or more spheres creates degeneracies. Figure 10 shows an example of a 3D grid with two fractures that are represented by a triangulation.

2.5 Improving intersections in 2D.

The algorithms introduced above for generating well and fracture sites are sufficient to give good grids as long as the individual well trajectories and/or fracture lines do not cross each other at sharp intersections. At such sharp intersections, additional care should be taken to ensure that the grid conforms in a feasible manner. There are three types of intersections a robust grid generator should handle: well–well, well–fracture, and fracture–fracture intersections. Our grid generator handles all these cases automatically, as well as harder cases such as the intersection of multiple fractures.

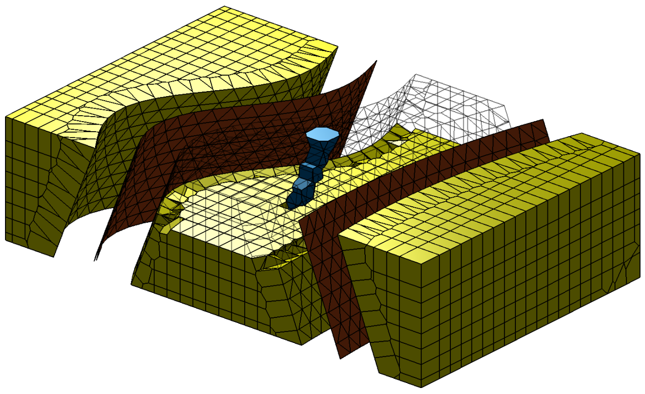


Fig. 10: Reservoir consisting of three blocks separated by fractures. Well cells are colored blue, while the lower dimensional grids of the fractures are shown as red surfaces.

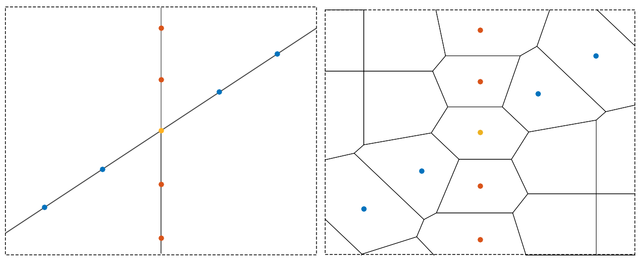

(a) The well trajectories and well sites. (b) A grid created using a Cartesian background-grid.

Fig. 11: Intersection of two well trajectories (black lines). Blue points are well sites for the diagonal well, red points are well sites for the vertical well, and the yellow point is a shared well site.

2.5.1 *Well–well intersections.*

When two well trajectories cross, we have to be careful when placing the well sites. If we place the sites of each well independently, consecutive sites will in general not be connected by Delaunay edges over the intersection. We may also create small and badly shaped cells. To treat these cases, all well paths are divided into segments by the well intersections. A well segment does not intersect any other well segments, except possibly at the end-points. When we place the well sites, we first place a well site at each intersection. A well site at an intersection is shared by all well segments starting or ending in this intersection. The remaining well sites are placed along the well segments as normal. Figure 11 shows the intersection of two wells. The yellow site is shared by both wells, and the other sites are in this case placed equidistant along the well curves. This method ensures a consistent size of the well cells, even at intersections of multiple wells.

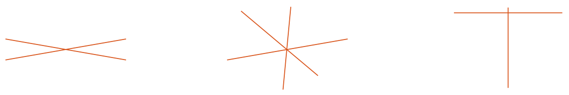

(a) (b) (c)

Fig. 12: Three hard cases to grid. (a) Fractures intersecting at sharp angles. (b) Multiple fractures intersecting. (b) Fractures that are barely intersecting.

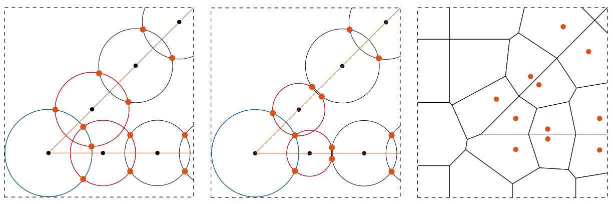

(a) Original sites. (b) Merged sites. (c) PEBI-grid

Fig. 13: Merging two conflict sites in a pinch-out. The orange lines are two intersecting fractures and red points the fracture sites.

2.5.2 Fracture–fracture intersections.

In a reservoir, it is common to have multiple fractures. Creating the fracture sites can then be much harder, as the fractures may intersect. The following algorithm handles the hard cases shown in Figure 12, which are common in a reservoir.

If we place the fracture sites for each fracture independently, we will in general not be able to represent the fractures exactly. At the intersection of two fractures, fracture sites from either fracture may interfere with each other and violate the fracture condition.

At each intersection, which corresponds to a 0D grid, we place a circle that is shared by all fractures ending in that intersection. The other circles are placed as normal along the fracture segments. We color all intersection circles blue, and all neighbor circles of blue circles are colored red. On each red circle, one of three actions is performed: *(i)* Nothing is modified, *(ii)* the radius is changed, *(iii)* the circle is merged with another red circle. If the interior of a circle does not contain any sites, it is not modified. If the interior of a circle contains a fracture site $\mathbf{f}_i$, we locate the red circle that generated $\mathbf{f}_i$. These two circles are tagged as conflict circles. The radii of the conflict circles are shrunk as shown in Figure 13. The new radii are chosen such that the blue circle and the two red circles intersect at the midpoint of the two fractures. When multiple fractures intersect, a circle might have multiple conflict pairs. We then calculate the new circle radius for each conflict pair and choose the smallest of them. If the radius of a red circle

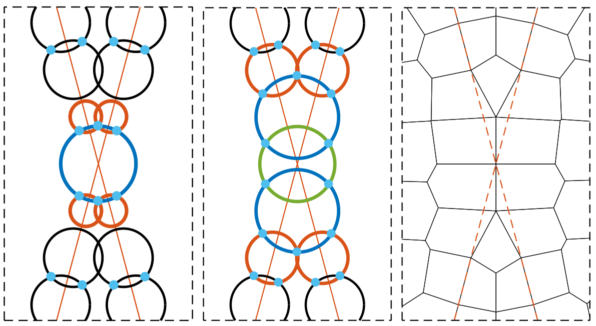

(a) (b) (c)

Fig. 14: The procedure of merging circles. (a) The circle at the intersection is colored blue, and its neighbors are colored red. The radii of the red circles are shrunk until they intersect the blue circle at the same point. The red circles now does not intersect with their neighbors and are therefore merged. (b) The merged circles are colored blue, and their neighbors are colored red. The green circle is already processed and is therefore not colored red. The procedure from (a) is repeated until the fracture condition is satisfied. (c) An associated grid.

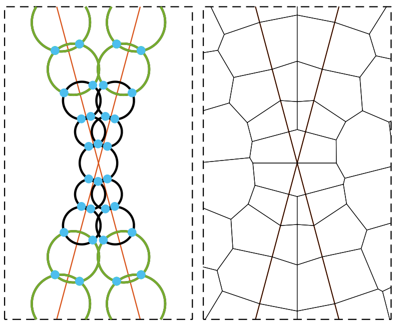

(a) (b)

Fig. 15: An alternative to the merging procedure in Figure 14. (a) Here two new 1D cells have been inserted for each fracture. The green circles are equal to those seen in Figure 14a. The refinement towards the fracture intersection is enough to avoid the problems of non-intersection circles. If the intersection had been sharper a larger refinement would have been needed. (b) An associated grid.

is shrunk too much, it might violate the radius condition of (1). For those cases, we locate the other red conflict circle sharing a fracture site with this circle. These two circles are merged to one circle centered at the midpoint of them. The merged circle is colored blue, and we repeat the procedure above. Figure 14 shows one iteration of the merging. In this case two sets of conflict circles are merged, one on each side of the intersection. This is enough to satisfy the fracture condition. If the

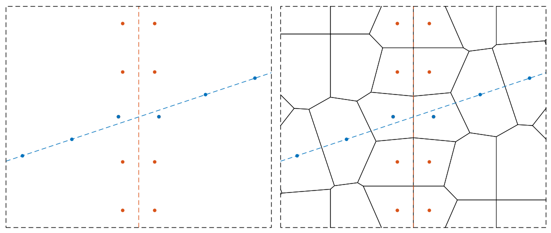

(a) Well and fracture sites. (b) PEBI-grid.

Fig. 16: Intersection of a fracture (red) and a well (blue). The fracture sites are placed as close as possible to the well trajectory. The fracture sites at the intersection are labeled as well sites.

intersection had been sharper, more than two pairs of circles might have been merged recursively.

An alternative to the merging of circles is to insert a new 1D cell between the two circles that do not intersect as shown in Figure 15. This is equivalent to a grid refinement of the 1D cells. This approach represent the fractures exactly at the cost of adding more cells.

Our method of splitting fractures into fracture segments and placing circle centers along these segments makes it easy to handle barely intersecting fractures. If a fracture segment is shorter than a specified length, we do not place any circles along it. In our implementation we have set this minimum length to be 80% of the desired length between circle centers.

2.5.3 Well–fracture intersections.

The last type of intersection we need to consider is well–fracture intersections. As for the two other cases, all fractures and wells are split at the intersections. Figure 16 shows the intersection of a well and a fracture. The first circle center of a fracture segment starting in a well–fracture intersection is placed half a step length from the start. Equivalently, the last circle center of a fracture segment ending in a well–fracture intersection is placed half a step length from the end. The two fracture sites created from the circle before and after the well–fracture intersection are labeled as well sites. These two sites are the first and last well site for the well segments starting and ending in the intersection, respectively.

In the case that well paths have higher priority than representing fractures, the construction can be modified so that the well sites are placed exactly on the well path, which will introduce a local deviation in the representation of the fracture surface.

2.6 Generating reservoir sites.

The reservoir sites can be placed any way that may fit the current problem as long as they do not violate the well and fracture conditions. The most obvious choice is to create a structured grid by placing sites equidistant in each direction or on a rectilinear mesh. We can then make use of the simplicity of the Cartesian topology away from the wells and fractures. When placing the reservoir sites, we ignore all fractures and wells. After the reservoir sites are created, we remove any sites violating the fracture or well condition. The resulting grid is then guaranteed to conform to fractures and wells. Some cells might still be small or badly shaped even if the fracture and well conditions are satisfied. We therefore also remove sites that are too close to each other. A grid size is defined for each well and fracture site. For well sites, the grid size is the distance between two consecutive well sites. For fracture sites, the grid size is set to the distance between the two sites that are generated by the same two circles. If a reservoir site is closer to a well or fracture site than that site's grid size, the reservoir site is removed. This ensures that the well- and fracture-cells have a consistent size.

There are many different methods to choose from to create a fully unstructured grid. Herein, we have chosen to place the reservoir sites using the force-based method proposed by Persson and Strang [32]. One reason is that this method is available as a free open-source implementation in MATLAB. The method associates the edges in the Delaunay triangulation with springs, whereas vertices are associated with joints connecting the springs. An initial triangulation is given, and the algorithm then finds an equilibrium position for the vertices. When solving for equilibrium, the well and fracture sites that have been created using the algorithms explained above are set as fixed points; that is, they are not allowed to move during the optimization procedure. For a detailed description of this method we refer the reader to Persson and Strang [32] or Appendix A.

As an alternative to the force-based method, we can use a similar algorithm to optimize the PEBI-grid directly instead of optimizing the dual Delaunay triangulation. We define the Centroidal PEBI-grid (CPG) *energy function* as [7, 17]

$$F(\mathbf{s}) = \sum_{i=1}^{n} \int_{\Omega_i \cap \Omega} |\mathbf{y} - \mathbf{x}_i|^2 \mathrm{d}\mathbf{y}.$$

The variable $\mathbf{s} = [\mathbf{s}_1^\top, \ldots, \mathbf{s}_n^\top]^\top$ is a vector of the PEBI-sites. The variable $\mathbf{x}_i$ is the mass centroid for PEBI-cell Ω_i. A necessary condition for F to be minimized is $\mathbf{s}_i = \mathbf{x}_i$, that is, the PEBI-sites coincide with the mass centroids [7]. The gradient of F is

$$\frac{\partial F}{\partial \mathbf{s}_i} = 2A_i(\mathbf{s}_i - \mathbf{x}_i),$$

where A_i is the area of the associated PEBI-cell. It was long thought that the energy function at most was continuous because of changes in topology when sites are moved. However, Liu et al. [22] proved that the energy function is two times differentiable for convex domains and almost always two times differentiable for non-convex domains. The exact Hessian is given explicitly [17, 22], and we can therefore use Newton's method to find the minimizer of the energy function. The computation of the Hessian is nonetheless expensive, and Liu et al. [22] showed the advantages of using quasi-Newton methods. Specifically, they show that the L-BFGS algorithm [29] performs better than both Newton's method and fixed-point iterations.

To be able to use the CPG formulation on a grid with fracture and well sites, we propose a small change to the gradient. The fracture and well sites are treated as fixed points, that is, we do not move them during the optimization procedure. We incorporate this into the L-BFGS algorithm by setting the derivatives $\frac{\partial F}{\partial \mathbf{s}_i}$ with respect to fixed points to zero. By doing so, the L-BFGS algorithm does not move the fixed points, and the resulting grid will conform to fractures and wells.

A comparison of the three methods for placing reservoir sites is shown in Figure 17. The two optimization methods have more uniform cells than the Cartesian background grid. Also, the cell centroids for the optimization methods are very close to the well trajectory.

Figure 18 summarizes the overall algorithm for generating a grid with control-point and boundary alignment for an optimal Delaunay background grid.

3 2.5D grids

It is not uncommon for an oil reservoir to have very large aspect ratios. An oil reservoir can stretch kilometers in the lateral directions, but only tens to hundred meters in the vertical direction. Reservoirs also have a natural and inherent layering since the rock is formed by a sedimentation process that creates horizontal or slightly inclined layers of deposits that are highly important to represent accurately in the grid. So-called 2.5D grids take advantage of the flexibility of 2D gridding to create unstructured tessellations in lateral direction, while retaining the simplicity of a Cartesian topology in the vertical direction. Such grids offer a relatively simple means of introducing local adaption with increased resolution laterally and have become very popular in parts of the industry [2], e.g., to accurately represent hydraulic fracturing around horizontal wells [37].

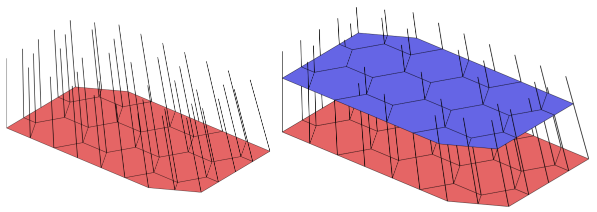


Fig. 19: The creation of a 2.5D grid. First, a 2D layer is gridded (red layer) and a set of pillars is placed through all vertices. Then, the grid is extruded along the pillars. Figures from Lie [21].

A 2.5D grid is created by generating a 2D areal tessellation and then projecting or extruding this to three-dimensions in the vertical direction, see e.g., [25] for a more extensive discussion. To this end, one starts by constructing a set of grid lines that each passes through a vertex in the areal grid. The grid is extruded along these lines, as shown in Figure 19, so that each cell in the lateral tessellation gives rise to a pillar of volumetric cells. By default, the pillars are vertical, but can also be set to follow major faults or other vertically inclined surfaces that need to be represented accurately. The hard part of creating a 2.5D grid is the choice of grid lines in the z-direction and the length each cell is extruded along these lines, as well as extruding vertically between multiple sets of areal tessellations.

4 Examples

In the following, we present a few examples to illustrate the capabilities of our gridding framework and the many types of adapted grids that can be generated.

Example 1 (Fracture networks) We start by two examples of areal grids that adapt to fracture networks. Inspired by hydraulic fracturing, we first create a grid of a reservoir with natural fractures and hydraulic fractures extending from a horizontal well, depicted in Figure 20a. The fracture and well sites are placed using the 2D algorithm, while the reservoir sites are placed by optimizing the dual Delaunay triangulation. The corresponding grid is shown in Figure 20b and includes refinement towards the fractures to obtain a higher resolution in these areas.

As a second illustration, we consider a statistical fracture network previously discussed by Shah et al. [34] (The data set is publicly available in the `hfm` module of MRST). The fracture system consists of 51 fracture

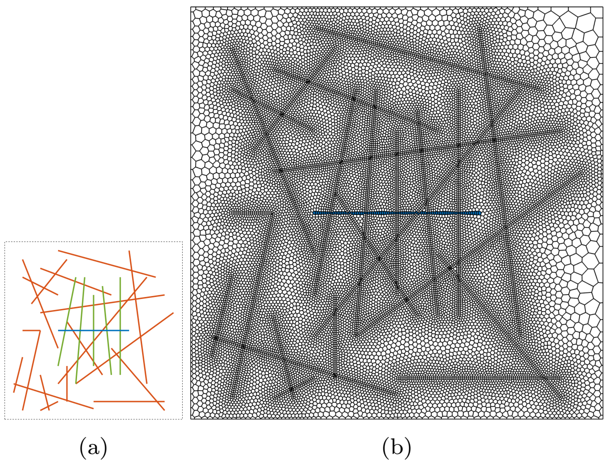


Fig. 20: Reservoir with hydraulic fractures. Natural fractures are shown in red, hydraulic fractures in green, and the blue line is a well. The right figure shows the corresponding grid for this reservoir, where the well cells have been colored blue.

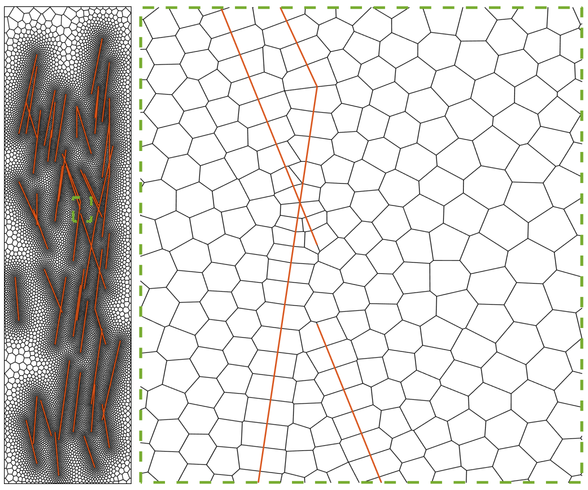


Fig. 21: Fractures generated from stochastic variables. Left figure shows the full reservoir, while the right figure shows a zoom-in of the green dashed square.

lines that have been generated by sampling stochastic variables defining the orientation, length and position for each fracture. Altogether, the lines form 31 disconnected fracture networks as shown in Figure 21. Stochastically generated fracture networks can be challenging to grid due to the common occurrence of small angles, fractures barely intersecting, and fractures arbitrary close, but not intersecting.

Both these examples show how our algorithm can create high quality grids of complex geometries with minimal effort, due to the automatic handling of intersections and local grid refinement.

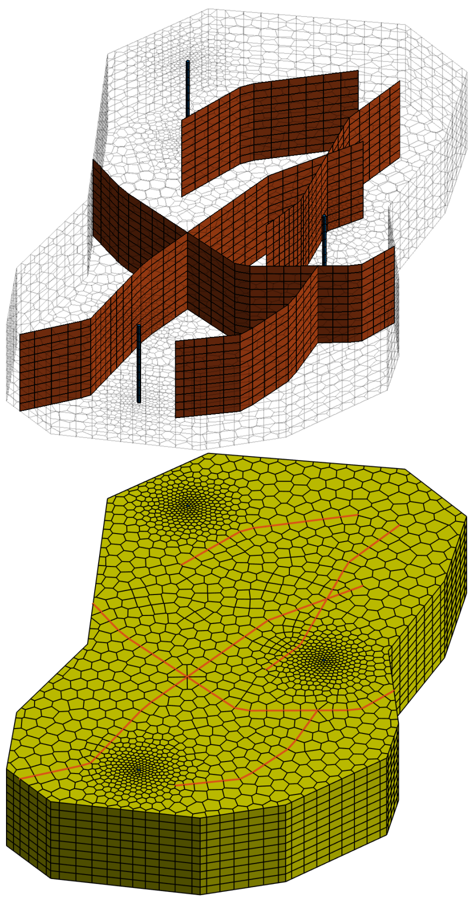


Fig. 22: A 2.5D grid of a fractured reservoir. The fracture faces and well cells in red and blue, respectively.

Example 2 (2.5D PEBI-grid) An important reason for having a robust 2D gridding algorithm is to be able to extrude the grids to form 2.5D volumetric grids. The grid shown in Figure 22 is generated by our 2D algorithm and then extruded to 3D along vertical lines. We have refined the cell size towards the wells, and this grid refinement is also respected by the 1D grids. In our implementation we set the radius of the circles around each vertex in the 1D grids as a function of the 1D cellsize, so that we in this way can automatically handle the local grid refinement also for cells along the fractures.

Example 3 (3D Voronoi grid with multilateral well) In the next example we show how our gridding algorithm can make a 3D Voronoi grid that conforms to a complex multilateral well. The well path depicted in Figure 23 curves horizontally in the domain and splits into multiple branches. The grid was made by first placing the well sites using our conforming algorithm. Reservoir sites were then subsequently optimized by use of the CPG algorithm.

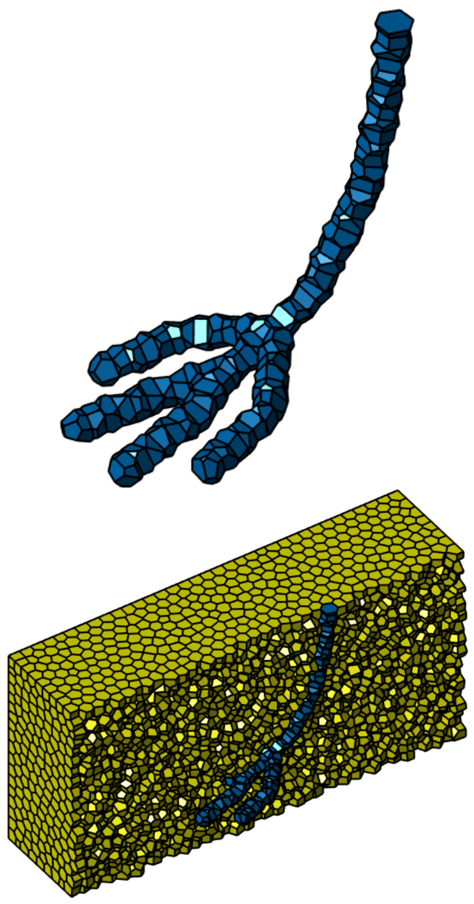


Fig. 23: A 3D grid with polyhedral Voronoi cells conforming to a well path.

Example 4 (3D fracture network) Figure 24 shows an example of a Voronoi grid conforming to a 3D fracture network. The fracture network consist of six fractures (2D constraints), 11 fracture intersections (1D constraints), and four intersections of intersections (0D constraints). The grid is generated by the general algorithm described in Section 2.2. The reservoir sites of the 1D grids are placed equidistant along the intersection lines. The reservoir sites of the 2D grids are place using the optimal Delaunay triangulation, while the reservoir sites of the 3D grid are placed using the CPG-gridding.

Building each grid separately enables us to choose the appropriate algorithm for creating the background grid, not only between dimensions, but also between two different grids of the same dimension. When generating the 2D grids, we are not able to use the improved techniques described in Section 2.4, because if we were to change the radius of a circle around a 1D vertex when creating a 2D grid, this radius has to be changed for all other 2D grids connected to this 1D vertex. As a result, some of the cells around 1D intersections are of lower quality. Implementing the required connection between 2D grids is out of the scope of this paper, and will be left for further work. In 2D, boundary vertices of the 1D intersections are captured exactly by adding an extra site. This has not yet been implemented in 3D. Faces representing fracture boundary cells thus are slightly larger than the prescribed fracture planes. Elsewhere, the faces, edges and vertices of the grid conform exactly to the lower-dimensional grids.

In some cases, one may want to consider the rock matrix as impermeable, and only run the fluid simulations on the fracture network. This is easily incorporated in our gridding algorithm by only constructing grids for the lower-dimensional constraints and not the 3D matrix volume.

5 Closing remarks

We have presented a method for generating PEBI-grids that conform to different geometric structures in subsurface reservoirs. Our method successfully creates both control-point aligned grids and boundary aligned grids in arbitrary dimensions. Our fracture condition gives a natural algorithm for creating boundary conforming grids in 2D and 3D. For the simpler case of 2D, we presented a more flexible algorithm with the following key advantages: (i) user-specified grid refinements allows for higher resolution in areas of interest; (ii) more robust handling of intersections; and (iii) high-quality cells even in constricted areas.

Through several examples, we have illustrated the flexibility and given indication of the robustness of our methods for generating 2D, 2.5D, and 3D grids. These examples show some of the possibilities of the algorithms presented in adapting to lower-dimensional constraints. Especially, the ability to create PEBI-grids that conform exactly to fractures in 3D is, to the best of our knowledge, a novel contribution to the literature.

Our hierarchical method for generating adapted 3D Voronoi grids is, in the way it has been presented herein, limited by the fact that constraints are only communicated upward from a lower-dimensional grid to a higher dimensional grid. A resulting issue is that there is generally no guarantee that sites of one fracture do not interfere with the fracture condition of another. This can happen if two fractures intersect at very sharp intersections, or lie close to each other, but do not intersect. We can resolve this issue to a large extent by refining the grid and our methods support the use of local grid refinement to avoid excessive number of cells. The algorithm can be improved by allowing for communication between different constraints of the same dimension so that the gridding of lower dimensional grids can be optimized with respect to each other. Herein, we have only presented this possibility in 2D, for the case of representing sharp fault or fracture intersection. In our experience, such a scheme improves grid quality significantly. We believe a similar approach is possible also in 3D, but have not yet implemented and tested it properly.

To what extent the methods presented above can be applied to efficiently mesh very large models is an open question. We believe that the algorithms presented herein should be relatively easy to include in existing PEBI meshing generators. By using a Cartesian background grid, you are only limited by how fast you can generate a PEBI-grid. In our MATLAB prototype implementation, the computational cost of creating fracture and well sites is low compared to the cost of generating the PEBI-grid. Likewise, the CPG algorithm is slow compared with the Delaunay triangulation optimization, since the latter utilizes fast libraries in MATLAB and only generates the PEBI-grid after the optimization has finished whereas the CPG algorithm has to create the grid at each iteration. Normally, both optimization algorithms give good grids in 20 to 50 iterations. One potential approach to reduce the associated computational cost could be to use a domain decomposition technique, but we have not yet researched this.

Another limiting factor is the ability to compute fracture intersections and intersections of intersections in an efficient and robust manner. Dense fracture networks often contain a large number of intricate and challenging special cases. The UPR module is still lacking somewhat in this respect.

The fracture condition from Definition 2 is sufficient and necessary for boundary conformity and enables us to check for conformity even before the grid is generated. The condition can also be used to locate (potential) areas of conflict between individual constraints such as sharp intersections before the actual grid is constructed, so that the user can adjust or prioritize the constrains or increase the local resolution to remedy potential gridding artifacts. The condition is quite simple and we believe it is an important tool, not only for the specific algorithms presented herein, but also for anyone who wishes to improve them or develop their own.

References

1. R. L. Berge. Unstructured pebi grids adapting to geological feautres in subsurface reservoirs. Master's thesis, Norwegian University of Science and Technology, 2016.
2. L. Branets, S. S. Ghai, S. L. Lyons, and X.-H. Wu. Efficient and accurate reservoir modeling using adaptive gridding with global scale up. In *Proceedings of the SPE Reservoir Simulation Symposium*, The Woodlands, Texas, Jan. 2009. doi: 10.2118/118946-MS.
3. L. V. Branets, S. S. Ghai, S. L. Lyons, and X.-H. Wu. Challenges and technologies in reservoir modeling. *Communications in Computational Physics*, 6(1):1–23, 2009.
4. M. Brewer, D. Camilleri, S. Ward, and T. Wong. Generation of hybrid grids for simulation of complex, unstructured reservoirs by a simulator with MPFA. *SPE Reservoir Simulation Symposium, 23-25 February, Houston, Texas, USA*, 2015. doi: 10.2118/173191-MS.
5. G. Courrioux et al. 3d volumetric modelling of cadomian terranes northern brittany, france): an automatic method using Voronoi diagrams. *Tectonophysics*, 331:181–196, 2001. doi: 10.1016/S0040-1951(00)00242-0.
6. X. Y. Ding and L. S. K. Fung. An unstructured gridding method for simulating faulted reservoirs populated with complex wells. In *Proceedings of the SPE Reservoir Simulation Symposium*, Houston, Texas, USA, February 2015. doi: 10.2118/173243-MS.
7. Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, 1999. doi: 10.1137/S0036144599352836.
8. D. D. Filippov et al. Reservoir modeling of complex structure reservoirs on dynamic adaptive 3D Pebi-grid. In *SPE Russian Petroleum Technology Conference, 16-18 October, Moscow, Russia*, 2017. doi: 10.2118/187799-MS.
9. L.-K. Fung, A. Hiebert, and L. Nghiem. Reservoir simulation with a control-volume finite-element method. *SPE Reservoir Engineering*, 7(3):349–356, Aug. 1992. doi: 10.2118/21224-PA.
10. L. S. K. Fung, X. Y. Ding, and A. H. Dogru. Unconstrained Voronoi grids for densely spaced complex wells in full-field reservoir simulation. *SPE Journal*, 19(5):803–815, Oct. 2014. doi: 10.2118/163648-PA.
11. S. Geiger-Boschung, S. K. Matthäi, J. Niessner, and R. Helmig. Black-oil simulations for three-component, three-phase flow in fractured porous media. *SPE Journal*, 14(02):338–354, 2009. doi: 10.2118/107485-PA.
12. E. J. Gringarten et al. New grids for robust reservoir modeling. In *SPE Annual Technical Conference and Exhibition, 21-24 September, Denver, Colorado, USA*. Society of Petroleum Engineers, 2008. doi: 10.2118/116649-MS.
13. E. J. Gringarten, M. A. Haouesse, G. B. Arpat, and L. X. Nghiem. Advantages of using vertical stair step faults in reservoir grids for flow simulation. In *SPE Reservoir Simulation Symposium, 2-4 February, The Woodlands, Texas*. Society of Petroleum Engineers, 2009. doi: 10.2118/119188-MS.
14. D. Guerillot and P. Swaby. An interactive 3D mesh builder for fluid flow reservoir simulation. *SPE Computer Applications*, 5(6):5–10, Nov. 1993. doi: 10.2118/26227-PA.
15. Z. E. Heinemann, C. W. Brand, M. Munka, and Y. M. Chen. Modeling reservoir geometry with irregular grids. *SPE Reservoir Engineering*, 6(2):225–232, May 1991. doi: 10.2118/18412-PA.
16. R. Holm, R. Kaufmann, B.-O. Heimsund, E. Øian, and M. S. Espedal. Meshing of domains with complex internal geometries. *Numerical Linear Algebra with Applications*, 13(9):717–731, 2006. ISSN 1099-1506. doi: 10.1002/nla.505.
17. M. Iri, K. Murota, and T. Ohya. *System Modelling and Optimization: Proceedings of the 11th IFIP Conference Copenhagen, Denmark, July 25–29, 1983*, A fast Voronoi-diagram algorithm with applications

to geographical optimization problems, pages 273–288. Springer Berlin Heidelberg, 1984. doi: 10.1007/BFb0008901.

18. M. Karimi-Fard, L. J. Durlofsky, and K. Aziz. An efficient discrete-fracture model applicable for general-purpose reservoir simulators. *SPE Journal*, 9(2):227–236, 2004. doi: 10.2118/88812-PA.
19. E. Keilegavlen, A. Fumagalli, R. Berge, I. Stefansson, and I. Berre. PorePy: An Open-Source Simulation Tool for Flow and Transport in Deformable Fractured Rocks. *ArXiv e-prints*, Dec. 2017.
20. Ø. S. Klemetsdal, R. L. Berge, K.-A. Lie, H. M. Nilsen, and O. Møyner. Unstructured gridding and consistent discretizations for reservoirs with faults and complex wells. In *Proceedings of the SPE Reservoir Simulation Symposium*, Jan. 2017. doi: 10.2118/182666-MS.
21. K.-A. Lie. *An Introduction to Reservoir Simulation Using MATLAB: User guide for the Matlab Reservoir Simulation Toolbox (MRST)*. SINTEF Digital, Dec 2016.
22. Y. Liu et al. On centroidal Voronoi tessellation-energy smoothness and fast computation. *ACM Trans. Graph.*, 28(4):101:1–101:17, Sept. 2009. `http://doi.acm.org/10.1145/1559755.1559758`.
23. B. Mallison et al. Unstructured Cut-cell grids for modeling complex reservoirs. *SPE Journal*, 19(02):340–352, 2014. doi: 10.2118/163642-PA.
24. S. Manzoor, M. G. Edwards, A. H. Dogru, and T. M. Al-Shaalan. Interior boundary-aligned unstructured grid generation and cell-centered versus vertex-centered CVD-MPFA performance. *Computational Geosciences*, Oct 2017. doi: 10.1007/s10596-017-9686-4.
25. X. Meng, Z. Duan, Q. Yang, and X. Liang. Local PEBI grid generation method for reverse faults. *Computers & Geosciences*, 110:73–80, 2018. doi: 10.1016/j.cageo.2017.09.012.
26. R. Merland, B. Lévy, and G. Caumon. Building PEBI grids conforming to 3D geological features using centroidal Voronoi tessellation. In R. Marschallinger and R. Zolb, editors, *Proceedings of IAMG*, page 12, Salzburg, 2011.
27. R. Merland, G. Caumon, B. Lévy, and P. Collon-Drouaillet. Voronoi grids conforming to 3D structural features. *Computational Geosciences*, 18(3-4):373–383, 2014. ISSN 1420-0597. doi: 10.1007/s10596-014-9408-0.
28. H. Mustapha. G23fm: a tool for meshing complex geological media. *Computational Geosciences*, 15(3):385–397, 2011. ISSN 1573-1499. doi: 10.1007/s10596-010-9210-6.
29. J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag New York, 2 edition, 2006. doi: 10.1007/978-0-387-40065-5.
30. C. Palagi and K. Aziz. Use of Voronoi grid in reservoir simulation. *Society of Petroleum Engineers*, 2(2):69–77, Apr. 1994. doi: 10.2118/22889-PA.
31. D. W. Peaceman. Interpretation of well-block pressures in numerical reservoir simulation. *Soc. Petrol. Eng. J.*, 18(3):183—194, 1978. doi: 10.2118/6893-PA.
32. P.-O. Persson and G. Strang. A Simple Mesh Generator in MATLAB. *SIAM Review*, 46(2):329–345, 2004. doi: 10.1137/S0036144503429121.
33. D. K. Ponting. Corner point geometry in reservoir simulation. In P. King, editor, *Proceedings of the 1st European Conference on Mathematics of Oil Recovery, Cambridge, 1989*, pages 45–65, Oxford, July 25–27 1989. Clarendon Press. doi: 10.3997/2214-4609.201411305.
34. S. Shah, O. Møyner, M. Tene, K.-A. Lie, and H. Hajibeygi. The multiscale restriction smoothed basis method for fractured porous media (F-MsRSB). *Journal of Computational Physics*, 318:36–57, 2016.
35. J. R. Shewchuk, S.-W. Cheng, and T. K. Dey. *Delaunay Mesh Generation*. Computer and Information Science. Chapman and Hall/CRC, 2012. doi: 10.1201/b12987-3.
36. J. Sun and D. Schechter. Optimization-based unstructured meshing algorithms for simulation of hydraulically and naturally fractured reservoirs with variable distribution of fracture aperture, spacing, length, and strike. *SPE Reservoir Evaluation & Engineering*, 18(04):463 – 480, November 2015. doi: 10.2118/170703-PA.
37. J. Sun, D. Schechter, and C.-K. Huang. Grid-sensitivity analysis and comparison between unstructured perpendicular bisector and structured tartan/local-grid-refinement grids for hydraulically fractured horizontal wells in eagle ford formation with complicated natural fractures. *SPE Journal*, 21(06):2–260, 2016. doi: 10.2118/177480-PA.
38. S. M. Toor, M. G. Edwards, A. H. Dogru, and T. M. Shaalan. Boundary aligned grid generation in three dimensions and CVD-MPFA discretization. In *Proceedings of the SPE Reservoir Simulation Symposium*, Houston, Texas, USA, feb 2015. doi: 10.2118/173313-MS.
39. S. Verma and K. Aziz. A control volume scheme for flexible grids in reservoir simulation. In *SPE Reservoir Simulation Symposium, 8-11 June, Dallas, Texas*. Society of Petroleum Engineers, 1997. doi: 10.2118/37999-MS.
40. X.-H. Wu and R. Parashkevov. Effect of grid deviation on flow solutions. *SPE Journal*, 14(01):67–77, March 2009. doi: 10.2118/92868-PA.
41. D.-M. Yan, W. Wang, B. Lévy, and Y. Liu. *Advances in Geometric Modeling and Processing: 6th International Conference, GMP 201 0, Castro Urdiales, Spain, June 16-18, 2010. Proceedings*, Efficient computation of 3D clipped Voronoi diagram, pp. 269–282. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13411-1_18.

A Optimal Delaunay triangulation

To create a fully unstructured grid, we can place the reservoir sites using the force-based method proposed by Persson and Strang [32]. For completeness, we will briefly review this method. The key idea is to associate edges in the Delaunay triangulation with springs, whereas vertices are associated with joints connecting the springs. The forces on each joint will depend on the difference between the actual length of the springs and their uncompressed length.

The uncompressed length l_0 of a spring is based on an element size function h. We evaluate the spring at its midpoint. For the domain $[0,1] \times [0,1]$ and element size function $h(x,y) = 1 + x$, the uncompressed length of the springs will be about twice as big in the right side of the domain as the left side.

We let the forces from the springs follow Hooke's law; that is, the force is proportional to the difference of its actual length l and its uncompressed length l_0. However, we assume that the springs only have repulsive forces, and no attractive forces. The force f from a spring is:

$$f(l, l_0) = \begin{cases} k(l_0 - l), & l < l_0, \\ 0, & l \geq l_0. \end{cases}$$

Here, k is a constant of value one that is needed to obtain the correct units.

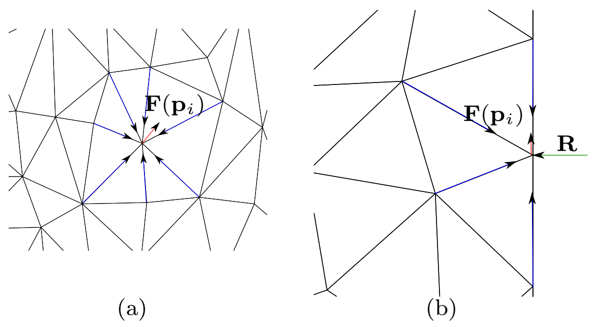
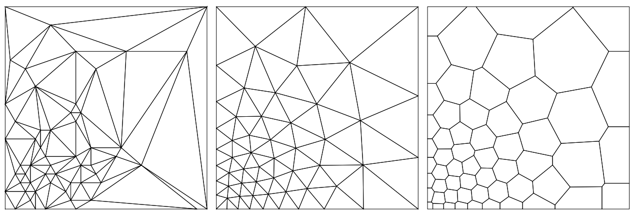


Fig. 25: Forces acting on a joint $\mathbf{p}_i$. Blue forces are the repulsive forces from each edge. The red force $\mathbf{F}(\mathbf{p}_i)$ is the sum of all repulsive forces. The lengths of the force vectors are not proportional to their magnitude. (a) An internal joint. (b) A joint on the boundary. An external force R is acting perpendicular to the boundary. The external force balance the internal forces so the joint will not move across the boundary.



(a) Initial Delaunay triangulation. (b) Triangulation after convergence. (c) Dual Voronoi diagram.

Fig. 26: Optimization of a triangulation using the force-based algorithm. The size of the elements is proportional to the distance from the origin squared $h(x,y) \sim x^2 + y^2$.

Let P be the coordinates of all joints. To find the force on a joint $\mathbf{p}_i$, we find the force from all springs connected to $\mathbf{p}_i$. The total force $\mathbf{F}(\mathbf{p}_i)$ is the sum of these forces. Figure 25a shows seven springs connected to one joint. The repulsive force from a spring acts in the longitudinal direction of the spring. We do not want the joints to move outside the domain we wish to triangulate. Figure 25b shows how an external force is added to the boundary joints. The external force is perpendicular to the boundary and balances the repulsive forces of the springs. Boundary joints can therefore only move along the boundary. We also allow for *fixed* joints thought of as glued to their initial position and not allowed to move, no matter how large the forces acting on them are.

The optimization loop of the force-based algorithm is very simple. We calculate the Delaunay triangulation of the joints P^k. For each edge in the triangulation, we calculate the repulsive force $f(l, l_0)$. For joints on the boundary we also add an external force to prevent it from passing over the boundary. The total force on a joint is found by summing all repulsive forces and external forces. The total force on a fixed joint is set to zero. All joints are moved a step length ξ along the direction of the total force acting on them:

$$\mathbf{p}_i^{k+1} = \mathbf{p}_i^k + \xi \mathbf{F}(\mathbf{p}_i^k).$$

An example of an optimum triangulation and its dual PEBI-grid is shown in Figure 26 for a case where initial reservoir sites were placed semi-randomly in the domain.

To achieve refinement towards wells, we create an element size function that decreases towards wells. We let the element size function decrease exponentially:

$$h_r(\mathbf{p}) = \min\left[h_{\max}, h_{\min} \exp\left(\frac{d(\mathbf{p}, W)}{\varepsilon}\right)\right]. \qquad (2)$$

The desired grid size of the background grid far from and close to the wells is $h_{\max}$ and $h_{\min}$ respectively. The distance $d(\mathbf{p}, W)$ is the closest distance from the point $\mathbf{p}$ to the set of well sites W. The constant ε controls the transition region. If ε is small, the refinement happens quickly around the wells. If ε is large, the transition region is large. When we run the force algorithm, all well and fracture sites are set as fixed points.

B Duality of Delaunay triangulation and PEBI-grids

There is a close relationship between the Delaunay triangulation and PEBI-grids. They are often called dual of each other in the sense that the topology of one is defined by the topology of the other. The duality is defined by a bijection between the faces of the Delaunay triangulation and the faces of the PEBI-grid. Following the presentation in Berge [1], we first define the k-face of a tessellation as a face of dimension k. In 2D a 2-face is a cell, a 1-face the edge between two cells and a 0-face a vertex. We then state the Voronoi-Delaunay duality precisely in the following theorem [35].

Theorem 1 (Duality of Delaunay triangulation and PEBI-grids)
Let P be a generic point set in $\mathbb{R}^d$. Let $\mathcal{V}$ and $\mathcal{T}$ be the associated PEBI-grid and Delaunay triangulation, respectively. Let $S = \{\mathbf{s}_1, \dots \mathbf{s}_j\} \subseteq P$ be a subset of the sites in P. The convex hull of S is a k-face of $\mathcal{T}$ if and only if $v_{s_1 \dots s_j}$ is a $(d-k)$-face of $\mathcal{V}$.

Proof First, assume that the convex hull of S is a k-face of $\mathcal{T}$. Then there exists a closed ball B that intersects $\mathbf{s}_1, \dots, \mathbf{s}_j$, but does not contain any sites from $P \setminus S$. The center of this ball is equidistant to all sites in S, hence, the intersection $v_{s_1 \dots s_j}$ is not empty; i.e., it is a PEBI-face of P. Let Π be the affine space that is orthogonal to the affine space of S and contains the center of B. The space Π has dimension $(d-k)$ because the dimension of $A(S)$ is k. All points in Π are equidistant to all sites in S, and no points in $\mathbb{R}^d \setminus \Pi$ are equidistant to all sites in S, thus, $v_{s_1 \dots s_j} \subseteq \Pi$. Let $0 < \varepsilon = \min_{\mathbf{p} \in P \setminus S} d(B, \mathbf{p})$ be the minimum distance from the ball B to any sites in $P \setminus S$. Any points in Π that are closer to the center of B than $\frac{1}{2}\varepsilon$ are on the face $v_{s_1 \dots s_j}$, hence, the dimension of $v_{s_1 \dots s_j}$ is the same as Π, that is $(d-k)$.

Now assume that $v_{s_1 \dots s_j}$ is a PEBI $(d-k)$-face. Since P is generic, there is no $s_{j+1} \in P \setminus S$ such that $v_{s_1 \dots s_j} = v_{s_1 \dots s_j s_{j+1}}$. In fact, the number of cells must equal $j = k + 1$ if $v_{s_1 \dots s_j}$ is to have dimension $(d-k)$. We can therefore find a closed ball centered at some point in $v_{s_1 \dots s_j}$ that has $s_1, \dots, s_j$ on its boundary and does not contain any sites $P \setminus S$. The convex hull of the $k+1$ sites in S is a k-simplex and it is strongly Delaunay, hence, it is a k-face in the Delaunay triangulation.

The main results of the duality theorem is for $j = 2$ and $j = d+1$. For $j = 2$ the theorem says that PEBI-cell v_{s_1} and v_{s_2} share a PEBI facet if and only if there is a Delaunay edge between site s_1 and s_2. For $j = d+1$ the theorem says that all PEBI-vertices are the center of a circumball of a Delaunay $(d+1)$-simplex. Figure 1 shows the duality in 2D.

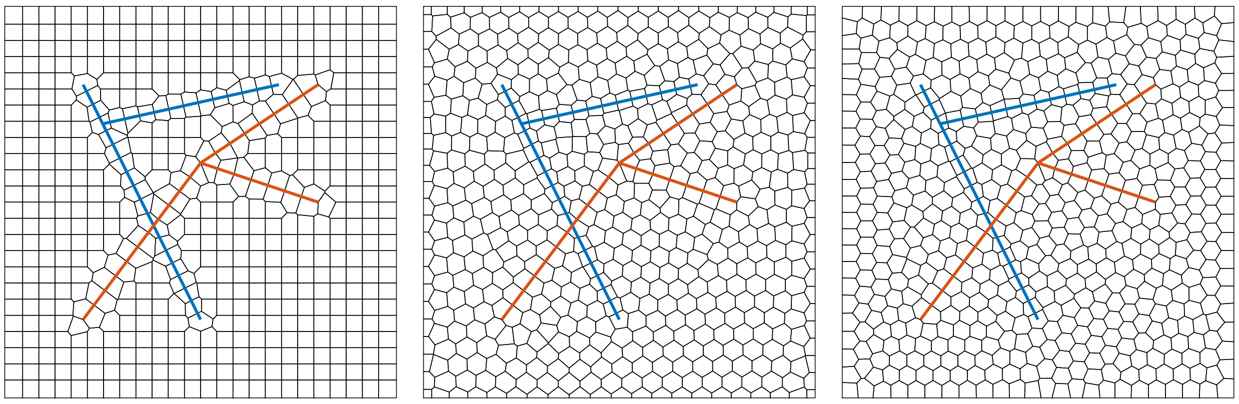
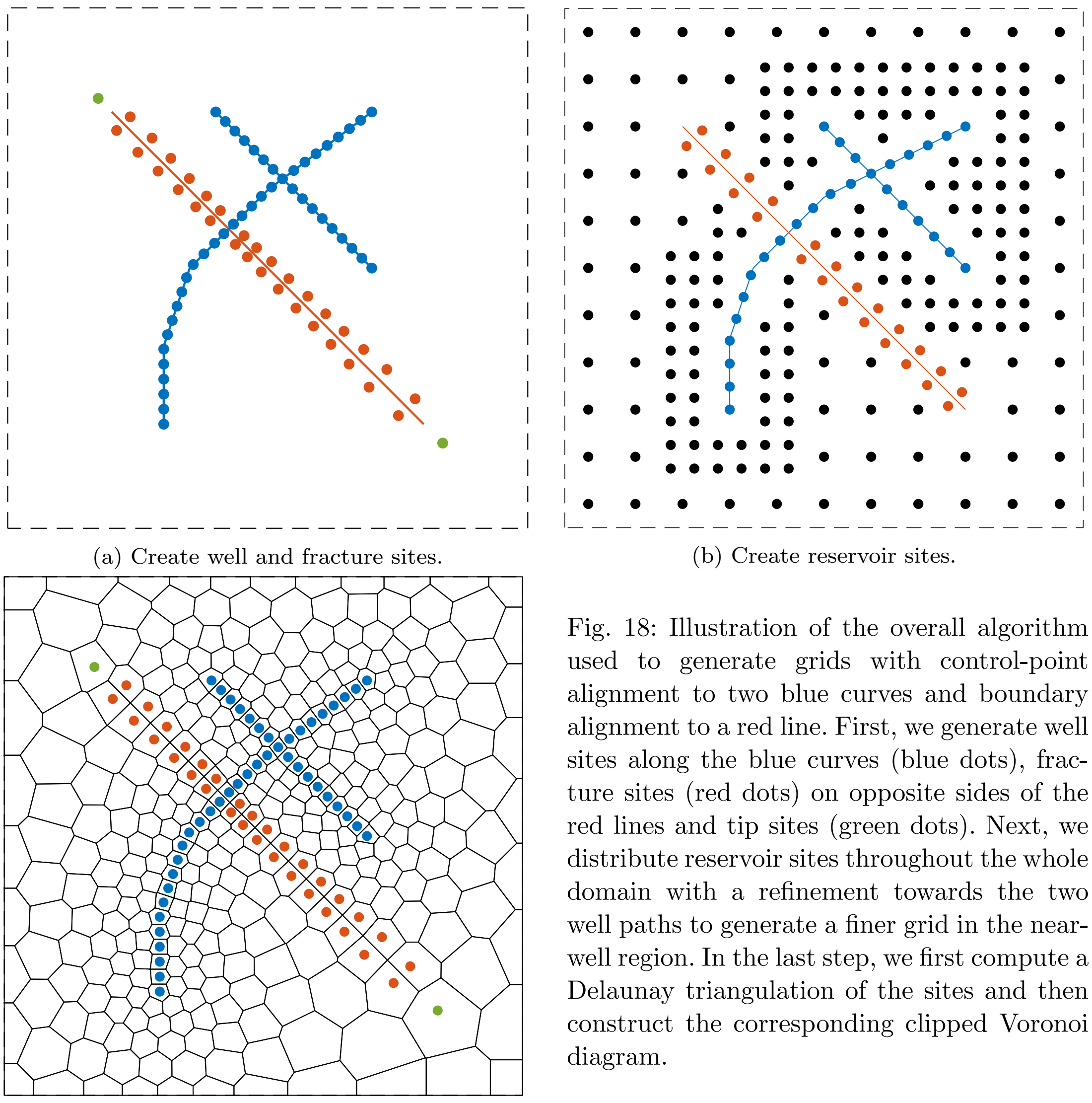

(a) Cartesian reservoir sites. (b) Optimized Delaunay. (c) Minimized CVD energy.

Fig. 17: Three grids of a reservoir. The reservoir has two wells (blue lines) and two fractures (orange lines). The well and fracture sites are the same for all three grids and are created using the methods described herein. The reservoir sites are created by three different methods: a Cartesian grid, optimizing the dual Delaunay triangulation, and minimizing the CVD energy function.


(a) Create well and fracture sites.

(b) Create reservoir sites.

(c) Associated PEBI-grid.

Fig. 18: Illustration of the overall algorithm used to generate grids with control-point alignment to two blue curves and boundary alignment to a red line. First, we generate well sites along the blue curves (blue dots), fracture sites (red dots) on opposite sides of the red lines and tip sites (green dots). Next, we distribute reservoir sites throughout the whole domain with a refinement towards the two well paths to generate a finer grid in the near-well region. In the last step, we first compute a Delaunay triangulation of the sites and then construct the corresponding clipped Voronoi diagram.

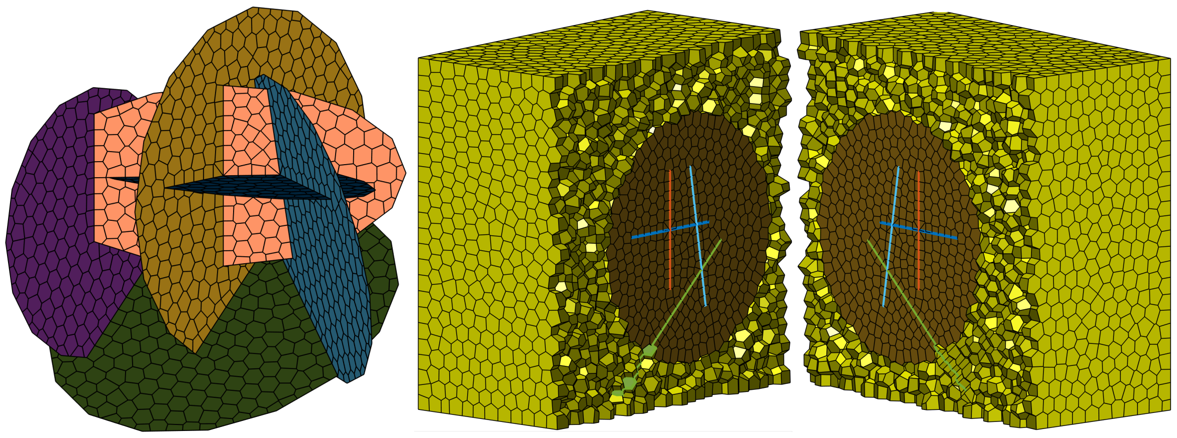


Fig. 24: A fracture network consisting of six fractures. The left figure shows the 2D grids of the fracture network. The right figure shows the 3D grid of the same network opened up along the plane of the large circular fracture in the middle.