

Mixed Element Trees: A Generalization of Modified Octrees for the Generation of Meshes for the Simulation of Complex 3-D Semiconductor Device Structures

Nancy Hitschfeld, Paolo Conti, and Wolfgang Fichtner

Abstract

This paper addresses the problem of the allocation of spatial grids for complex non-planar three-dimensional (3-D) semiconductor device structures. We have characterized the class of meshes suitable for the integration of the device equations with the usual numerical schemes as being a subclass of the class of Delaunay meshes. We propose an algorithm for the efficient generation of such admissible meshes based on the iterative refinement of coarse elements. The generated meshes permit an exact geometrical modeling of rather general domain boundaries of modern silicon devices avoiding the “obtuse angle problem” by construction.

1 Introduction

Software packages for the numerical solution of partial differential equations (PDEs) are important prototyping tools in several engineering disciplines. Results from numerical simulations are used to design and to optimize parts ranging from large aircraft wings to microscopic semiconductor devices and integrated circuits.

The spatial discretization of the structure to be simulated, i.e. its subdivision in cells, is key to the accuracy of the computed solution. An appropriate mesh should fulfill several requirements: first, it must provide a reasonable approximation of the geometry to be modeled, in particular of its boundary and internal material interfaces. Second, it is extremely important to accurately approximate all internal quantities relevant to the solution of the PDEs, such as the doping profile in a semiconductor device. Third, the single cells must fulfill certain geometric constraints imposed by the numerical integration method: if the PDEs are solved with the finite element method, no angle must be smaller than some bound supplied *a priori*. However, if the equations are solved using a control volume discretization or box method (BM)[1], the surfaces defining the volume discretization have perpendicular bisectors of the edges of the tessellation[2]. It will be shown that a subclass of the class of Delaunay tessellations satisfies this requirement.

The methods used to obtain tessellations satisfying the above requirements can be roughly separated into two classes.

- *A posteriori* tessellation of a given point set: An appropriate number of points is distributed on the boundary and in the interior of the integration domain. Subsequently, edges coupling neighbour points are introduced in order to obtain a proper finite element mesh; typically, a Delaunay triangulation of the given point set is constructed [3].
- Iterative refinement of coarse elements: An initial coarse mesh is allocated first. The elements that do not appropriately approximate the domain boundary or the internal

quantities are recursively subdivided into smaller elements, typically through the addition of new grid points on their edge midpoints (bisection-based algorithms)[4].

Iterative refinement of coarse elements presents several advantages compared to the *a posteriori* tessellation of a given point set. First, the distribution of a point set with appropriate point density is usually a difficult task. Conversely, internal quantities are sometimes known *a priori* for the vertices generated so far, and therefore it can be decided whether an element is small enough or whether it should be subdivided further. In addition, bisection-based algorithms are well-suited for incremental grid adaptation based on an estimate of the error in the solution in each element. Quadtrees and octrees are the most well-known implementation for iterative refinement in 2-D and 3-D, respectively (See references below).

In two dimensions(2-D), both rectangles and triangles have been used for the initial coverage of the integration domain. Bank and co-workers have proposed covering the integration domain (a closed polygon) with a carefully chosen triangulation, and to recursively subdivide triangles into four similar triangles through the addition of four new edge midpoints [5, 6]. Yerry and Shephard proposed using a modified quadtree data structure: the integration domain is encapsulated in a square, and the square's quadrants are recursively subdivided until the mesh density is sufficient to model the domain geometry and internal quantities appropriately [7]. Müller et al.[8] have recently extended their idea in the implementation of a fully automatic 2-D mesh generator.

In three dimensions(3-D), the recursive refinement of simplexes (tetrahedra in this case) is much harder than in 2-D[9]. Major problems include the difficulty to generate a well-shaped initial tetrahedral grid, the impossibility to regularly subdivide tetrahedra in similar sub-elements, and the difficulties arising when tetrahedra lying between dense and coarse mesh regions must be subdivided in proper cells. The modified quadtree approach on the other hand, has been successfully extended to three dimensions [10, 11, 12]. The 3-D domain is enclosed in a cuboid, whose octants are repeatedly refined until the boundary and internal quantities are sufficiently approximated. A slightly different approach for the generation of octree-based Delaunay meshes has been proposed by Schroeder and Shephard[4].

The grid generator Ω presented in this paper has been developed for the simulation of semiconductor devices in 3-D. Several aspects must be considered when tessellating 3-D semiconductor devices. As these devices feature rather complex geometries the grid generator must be capable of dealing with quite general domain boundaries and material interfaces. In the simulation of mechanical parts similar problems arise. Because classical finite element schemes seem inappropriate, PDEs are usually solved using the control volume or box method [1]. This leads to the requirement that the underlying mesh should be a Delaunay grid[13]. In addition, in a semiconductor device featuring sizes of several μm , relevant internal quantities may vary over several orders of magnitude within 100 Å, i.e. within a few thousandths of the overall size. Hence, mesh density must vary over several orders of magnitude in the whole device.

The first version of the grid generator, Ω_{oct} (Ω octree), as presented in [2], was based on a conventional modified octree approach. It was shown that the octants of an octree could be completed with additional edges on the surface or in the interior of each octant. However, in order to obtain an appropriate mesh, modified octrees suffer three serious drawbacks when used for discretizing semiconductor devices:

1. All octants are almost cubic; therefore the point density is locally constant along all three coordinate axes. However, relevant quantities in a semiconductor device may strongly vary along one axis, while remaining constant in the plane perpendicular to the axis. In

these cases, isotropic cells, as those resulting from modified octrees, lead to a large number of redundant mesh points.

2. The algorithm used to obtain Delaunay meshes requires that material interfaces intersect an octant at its edge midpoints. As a result, several refinements and a huge number of mesh points were required along material interfaces.
3. There may be inaccuracies in computing the volume discretization for interface elements.

In order to overcome these problems, the current version $\Omega_{\text{me}}(\Omega$ with mixed elements) generalizes the modified octree approach in several aspects. The whole device is no longer encapsulated in a single octree, but partitioned in a set of *macro-elements* consisting of cubes, rectangular prisms and rectangular pyramids. This approach permits the representation of arbitrary plane-faced geometries using a reasonable number of mesh points. No refinement along material interfaces is required to fit the geometry of the device. Subsequently, the desired mesh density in the interior of the device is obtained through the recursive refinement of prisms, pyramids and cuboids (bisected-based algorithm). If a finer mesh is required along one, two, or three coordinate axes, cubes, for example, are subdivided into two halves, four quadrants, and eight octants, respectively. Then, the elements with additional edge midpoints are subdivided into tetrahedra, pyramids, prisms or bricks in order to get a proper finite element mesh.

The rest of this paper is organized as follows: Section 2 recalls the definitions of Delaunay tessellations and Voronoi faces and describes their relation with the BM. Section 3 characterizes the set of elements used to fit the device geometry, the refinement process and also the restrictions imposed on irregular leaves in order to obtain proper meshes. It is shown that meshes generated with Ω_{me} are by construction a subclass of Delaunay meshes. Section 4 presents a general outline of the grid generation process and provides details about the data structures and algorithms implemented. Section 5 shows examples of meshes generated for the simulation of geometrically complex semiconductor devices. Section 6 gives information about the time and space requirements of Ω_{me} and compares these requirements with those of the modified octree version Ω_{oct} . Section 7 contains our conclusions.

2 Basic Definitions

The next definitions establish the relation between the BM and Delaunay meshes. In the following, let S be a set of N points $\{p_0, \dots, p_{N-1} | p_i \in \mathbb{R}^d\}$, and let C be the convex hull of S .

Definition 1 *The Voronoi polytope or Voronoi region associated with each point p_i is the locus of points that are closer to p_i than to any other point of S . For any point set S , the Voronoi polytopes partition the plane or space into a convex net, called the Voronoi diagram of S , denoted as $V(S)$. The vertices, edges and faces of the net are called Voronoi points, edges and faces.*

Clearly, the Voronoi region of $p_i \in S$ is equal to the intersection of all the half-spaces determined by the mid-perpendiculars of all segments $p_i p_j$, $p_i \neq p_j$. Voronoi faces always lie in such perpendicular bisectors.

The interest of Voronoi polytopes for the numerical integration of PDEs is due to the following Theorem (Delaunay [14]):

Theorem 1 (Delaunay) *Let S be a point set in \mathbb{R}^d , $V(S)$ be the Voronoi diagram of S . Then, the straight line dual¹ of $V(S)$ is a tessellation of the convex hull of S .*

If the Voronoi boxes are to be used for the discretization of PDEs with the BM, each edge perpendicular to a Voronoi cross section should be an edge of the finite element grid. The following definitions characterize the tessellations suited for the integration of PDEs with the BM.

Definition 2 *An edge $p_i p_j$ is a Delaunay edge if there exists a non-degenerate Voronoi cross section in $V(S)$ lying in the mid-perpendicular of $p_i p_j$.*

Definition 3 *A tessellation T of S is a Delaunay tessellation if all Delaunay edges of S are edges of the tessellation T .*

Theorems 2, 3 and 4 characterize Delaunay tessellations. They will be used in Section 3 to show that meshes generated using Ω_{me} are subclass of Delaunay meshes appropriate for the BM.

Theorem 2 *An edge $p_i p_j$ is a Delaunay edge of S if and only if there exists a point-free circumsphere of $p_i p_j$ ² [13].*

Theorem 3 *In 3-D, a face f_{ijk} is a Delaunay face of S if and only if there exists a point-free circumsphere of f_{ijk} [13].*

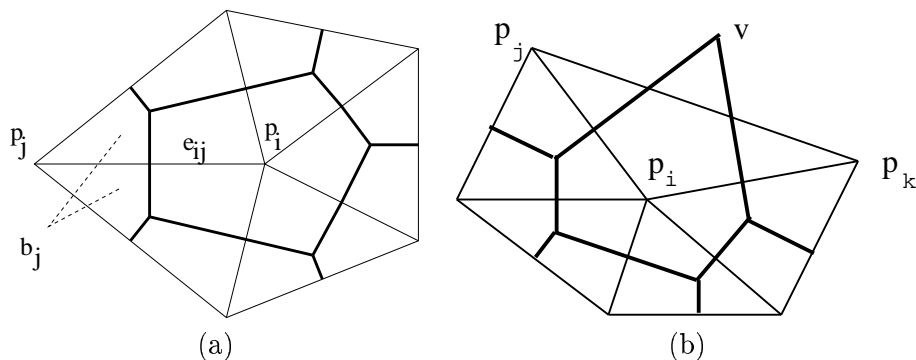


Figure 1: 2-D Delaunay tessellations and their Voronoi Diagrams. The tessellation shown in (a) is proper for the BM, but the one shown in (b) is not.

According to Theorem 2, the two tessellations shown in Fig. 1 are Delaunay tessellations (the thick lines correspond to the Voronoi cross sections, and the thin lines are Delaunay edges). In Fig. 1(a), Voronoi regions correspond to the numerical area integration around each point: for example, the Voronoi region b_j is associated with p_j . The same does not hold for the tessellation shown in Fig. 1(b). The Voronoi region around p_i includes a part outside of the convex hull because the Voronoi point v lies outside of the tessellation, and the area integration is larger than it should be.

The area integration (volume integration in 3-D) cannot be properly computed if Voronoi points lie outside of the convex hull. The following theorem characterizes such tessellations.

¹The straight line dual is defined as follows: the edge $p_i p_j$ is part of the tessellation if there is a non-degenerate Voronoi face lying in $p_i p_j$.

²A point-free circumsphere of $p_i p_j$ is a closed hypersphere containing p_i and p_j but no other points of S .

Theorem 4 *Let $S \subset \mathbb{R}^n$, $n \leq 3$ be a set of points, C the convex hull of S , and T a Delaunay tessellation of S . Then no Voronoi point of S lies outside C if and only if for each edge $p_i p_j$ (face f_{ijk} in 3-D) of T on the surface of C , the circumcircle of $p_i p_j$ (circumsphere of f_{ijk}) with the center in the middle of $p_i p_j$ (f_{ijk}) is point-free.*

Proof: Consider Fig. 2: the centers of all circumcircles of edge $p_i p_j$ lie on the mid-perpendicular

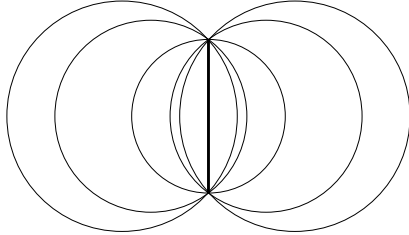


Figure 2: Circumcircles of edge $p_i p_j$

of $p_i p_j$. If the center of the circle moves from left to right, the part of the new circle left to $p_i p_j$ is included in the old circle, while the part right to $p_i p_j$ includes the corresponding part of the old circle. Obviously, a similar result holds for 3-D.

Now suppose there is a Voronoi point p_v outside C , which is the center of the circumsphere of the element el_{p_v} of T dual to p_v . Face f_{ijk} of this element lies on the surface of C . If the center of the circumsphere of f_{ijk} is moved to the center of f_{ijk} , the new sphere includes all vertices of el_{p_v} since these points lie “on the other side” of f_{ijk} . Hence, the new circumsphere of f_{ijk} is not point-free.

Conversely, suppose there is a face f_{ijk} lying on the surface of C whose circumsphere with the center in its midpoint is not point free. The element el containing this face has a vertex which is closer to the face midpoint than the face points. The center p_v of the element el is at the other side of this face outside of the convex hull C . As p_v is also the Voronoi point where some midperpendicular cross sections to the edges of the element el join, there exists a Voronoi Point outside of the convex hull. \square In 2-d, the proof is analogous.

3 Characteristics of Well-shaped Meshes for 3-D Structures

In this section, we characterize finite element meshes based on mixed element trees and show that they are well-shaped³ by construction. This characterization includes a formal definition of a well-shaped tessellation (Section 3.1), and the properties that elements have to satisfy. Elements used to fit the device geometry (also called *macro-elements*) are described in Section 3.2. The required mesh density is obtained using an iterative refinement of coarse elements. Section 3.3 shows the different ways to refine the macro-elements. Finally, elements with bisected edges have to be tessellated into a set of valid 3-D elements. The feasibility of a proper tessellation depends on the eccentricity of the element (Section 3.4). In Section 3.5, the concept of a mixed element tree is introduced and we show that mixed element trees generate *well-shaped* meshes by construction(Section 3.5).

³A mesh is called *well-shaped* if the BM can be applied.

3.1 Definition of a Well-shaped Mesh

The geometry of a 3-D structure is described using a set of general polyhedra that represent regions of different materials. The faces of each polyhedron define part of the boundary or an interface that have to be respected by the tessellation.

The characteristics of a discretization inside of a polyhedron proper for the BM are described in the following definition:

Definition 4 *Let P be a general polyhedron. A tessellation T of P is well-shaped if*

- (i) *T is a Delaunay tessellation,*
- (ii) *no Voronoi Point lies outside of T .*

Condition(i) guarantees that each existent Voronoi face is associated with an edge that belongs to the tessellation(Definition 3). Condition(ii) guarantees that the Voronoi region associated with each point corresponds to the volume discretization required by the BM, that means, the integration around each point can be done using just the Voronoi faces(See Fig. 1).

3.2 Proper Set of Elements

The following theorem characterizes the set macro-elements used in the generation of well-shaped meshes.

Theorem 5 *Let P be a set of polyhedra. P leads to well-shaped meshes if each polyhedron $p \in P$ can be*

- (i) *discretized using the BM, and*
- (ii) *refined in such a way that all newly generated polyhedra also belong to P (P is closed).*

Proof: If each element satisfies condition (i), perpendicular cross sections (Voronoi faces) can be associated with each element edge in order to properly compute a local integration of the volume around each element point. Condition (ii) guarantees that for each element generated through the refinement process it will be possible to fulfill condition (i). \square

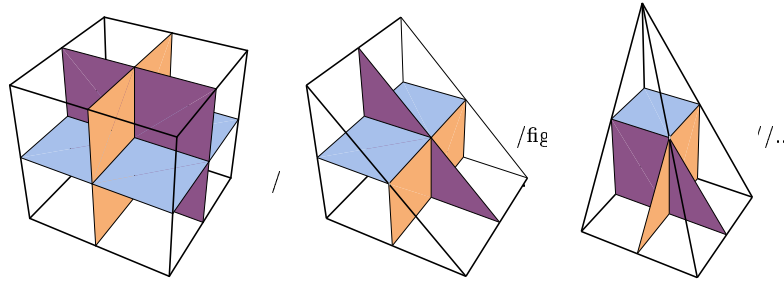


Figure 3: Voronoi faces for bricks, prisms and pyramids

Our set of macro-elements is composed of rectangular pyramids, rectangular prisms and bricks. In Fig. 3, we show the proper volume discretization of a brick, a prism and a pyramid. As will be shown in Section 3.3, these element types also satisfy condition (ii) of Theorem 5.

The rectangular tetrahedron cannot be used as macro-element because it is not possible to fulfill condition (i) of Theorem 5. The Voronoi point p lies outside of the element as shown in the

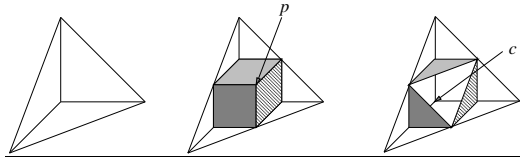


Figure 4: Rectangular tetrahedron and its Voronoi surfaces

center tetrahedron of Fig. 4. If the part of the Voronoi surfaces lying outside of the elements were cut as shown in the right tetrahedron, the remaining parts would not be sufficient to compute the volume integration around the vertex c .

3.3 Regular Refinement of Macro-elements

In order to simplify the grid generation process, macro-elements considered too coarse are split along the edge midpoints.

3.3.1 Regular Refinement of Cuboids

Cuboids are split into two halves, four quarters or eight octants as shown in Fig. 5.

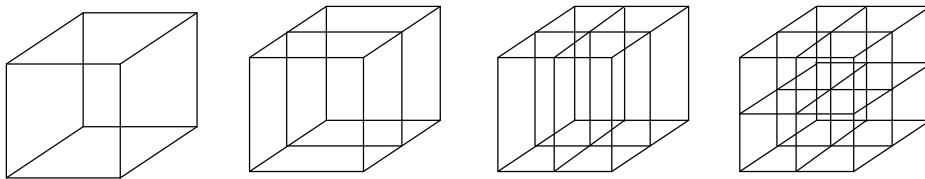


Figure 5: Cuboids refined in one, two or three directions generate two, four and eight cuboids, respectively.

3.3.2 Regular Refinement of Prisms

Rectangular prisms are split either across the axis perpendicular to their triangular faces, across both axes lying in the triangular faces, or across all three axes. Fig. 6 shows the different ways to split a prism: the left prism is a rectangular prism without split edges, the next one is split across the axis perpendicular to the triangular faces; it decays into two smaller prisms. The third prism is split across the two axes spanning the triangular faces; it decays into two prisms with triangular faces similar to those of the original prism, and into a cuboid. Finally, the last prism is split along its three axes; so it decays into four prisms similar to the original prism, and into two cuboids. Notice that each cuboid could be split into two prisms similar to the original one, but since cuboids offer more flexibility for further refinement it is preferable to keep them intact.

3.3.3 Regular Refinement of Pyramids

Once all edges of a rectangular pyramid have been split, a pyramid decays into a cuboid, two pyramids similar to the original one, and two rectangular prisms (Fig. 7). Rectangular pyramids

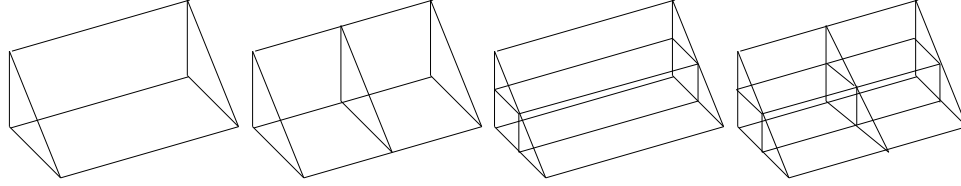


Figure 6: Prism refinement in one, two or three directions generates two prisms, one cuboid plus two prisms, and two cuboids plus four prisms, respectively.

cannot be refined across only one or two axes, since the corresponding partitions lead to 3-D elements that violate the requirements of Theorem 5. On the other hand, pyramids never occur when cuboids and prisms are refined, and only two of them are generated when a pyramid is refined. As a result, the relative number of pyramids tends to decrease when elements are refined, and the number of cuboids and prisms, whose splitting can be finely tuned, tends to increase accordingly.

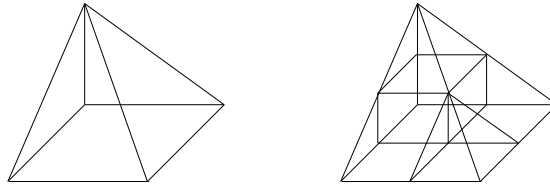


Figure 7: Pyramid refinement in the three directions generates one cuboid, two prisms and two pyramids.

3.4 Manipulation of Irregular Leaves

Definition 5 *Let l be a macro-element. l is called 1-irregular if all of its edges are bisected at most once. l is called regular if non of its edges is bisected.*

Irregular macro-elements appear after refining the grid elements according to given density requirements. Irregular macro-elements, i.e., elements with edges bisected more than once are made 1-irregular looking for well-shaped tessellations. The next definition characterizes such an element.

Definition 6 *Let l be a 1-irregular macro-element. l is a well-shaped if no Voronoi point of l lies outside its convex hull (in this case, the macro-element itself).*

The previous definition can be fulfilled by finding a Delaunay tessellation of the 1-irregular macro-element that satisfies Theorem 4. (The set of points S is defined by the element points and edge midpoints.) As shown in Fig. 8 using a 2-D example, not all 1-irregular elements are well-shaped 1-irregular elements. Circumcircles fulfilling Theorem 4 can be found for each edge of the left triangle if $a > b$ and if a new point is inserted as shown in the right triangle.

General lower bounds for the brick's eccentricity depending only on the existence of face midpoints and edge midpoints have been found. They are summarized in the following Theorem and Corollary[13]:

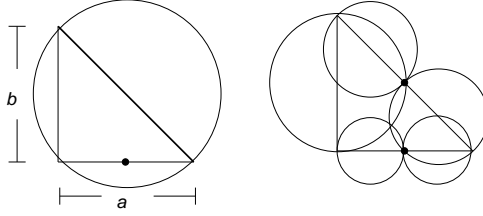


Figure 8: No point-free circle for the diagonal edge of the left triangle(left) and point-free circles for all the edges of the right triangle

Theorem 6 *Let l be a 1-irregular brick with edge lengths a, b, c . Assume that no face f_{ab} of size $a \times b$ carries a mid-face point if $2c \leq \sqrt{a^2 + b^2}$ and no edge e_a carries a mid-edge point if $2b \leq a$, or $2c \leq a$. Then l is well-shaped.*

Corollary 7 *Let T be a 1-irregular brick leaf with edge lengths a, b, c , no edge e_a is split unless $a < \sqrt{2}b$ and $a < \sqrt{2}c$. Then, l is well-shaped.*

Similar bounds cannot be provided for pyramids and prisms due to limitations imposed in handling their 1-irregular triangular faces. Consider, for example, Fig. 8: if a prism has the left 1-irregular triangle shown in this figure as one of its triangle faces, point-free circumspheres for all of the boundary faces fulfilling Theorem 4 cannot be found. However, eccentricity⁴ requirements satisfying Theorem 4 for each particular 1-irregular element have been determined and built into Ω_{me} .

3.5 Well-shaped Meshes using a Mixed Element Tree

The tessellation of a device is represented using a mixed element tree. This representation allows to generate well-shaped meshes by construction because it permits to implement the concepts introduced above in a natural way.

Definition 7 *Let T be a tree. T is a mixed element tree if*

- (i) *each node (internal or leaf) is a polyhedron*
- (ii) *each internal node is labeled with the axes across which the node is refined.*

Theorem 8 *Let T be a mixed element tree. T leads to well-shaped meshes by construction if and only if*

- (i) *each internal node is one of the macro-elements described in Section 3.2,*
- (ii) *each leaf is an irregular macro-element satisfying Definition 6 or a regular element, and*
- (iii) *each 1-irregular leaf allows its tessellation into tetrahedra⁵, pyramids, prisms and bricks (final element mesh).*

Proof: Condition(i) guarantees that no Voronoi point lies outside of a boundary and interfaces. As each internal node is refined independently, no Voronoi point must also lie outside of

⁴The eccentricity of an element is the ratio between the lengths of the longest and shortest edge incident at the element.

⁵Notice that a tetrahedron can be used as a valid element of a tessellation because Voronoi faces have to be inside of the 1-irregular leaf but not necessarily inside of each element of the tessellation

any 1-irregular macro-element. This criterion is fulfilled using condition(ii). As a result, a grid composed of elements satisfying these properties is well-shaped by construction because each edge within each element can be associated with a Voronoi face . For edges belonging to several neighbor elements, the Voronoi face is given by the sum of the perpendicular cross sections inside of each macro-element. The resulting grid edges are a Delaunay tessellation because they satisfy Definition 2. Finally, condition (iii) defines the elements used to tessellate 1-irregular elements and so the final grid is obtained. \square

4 The Mesh Generation Process

The complete grid generation process can be outlined in Algorithm 1.

```

Read_Device_Geometry(geometry_description);
Read_Refinement_and_Doping_Data(refinement_data, doping_data);
Fit_Device_Geometry(grid, geometry_description);
/* An initial grid composed of bricks, rectangular prisms and rectangular
pyramids is generated. The grid is handled as a forest where each
macro-element is the root of a tree. */
Refine_Grid(grid, refinement_data, doping_data);
/* An irregular grid is generated refining each macro-element independently
of the others, in order to fit physical and other geometrical parameters. */
Make_Irregular_Leaves_Splittable(grid);
/* A finite element grid is obtained after tessellating all the irregular
elements into tetrahedra, pyramids, prism and bricks if the Voronoi cross
sections inside of each irregular element can be computed. */
Assemble_Voronoi_Surfaces(grid);
Compute_Regions(grid, geometry_description, region_information);
/* Grid elements are separated in regions according to input parameters. */
Write_Geometrical_Information(grid, region_information);
/* Point coordinates, points per element and Voronoi surfaces are
stored in a file. */
Write_Doping_Information(grid, doping_data)
/* Doping values for each grid point are stored in a file. */

```

Algorithm 1: Grid generation process

4.1 Global Data Structures

4.1.1 Description of a mixed Element Tree Node

In order to store the necessary information to handle this kind of tree, we have defined the data structure shown in Fig. 9.

The `type` field indicates the element type, i.e. cuboid, prism or pyramid. The `orientation` array stores how the current element is oriented with respect to the global, right-hand axes coordinate system. This information is computed for all macro-elements of the initial grid and updated during element refinement. Normally, each son inherits the orientation from his father. However, in the partition of a pyramid one prism has another orientation. `split_axes` identifies the local axes across the node has been split. The array `refinement_levels` permits

the computation of the depth of the forest. The depth of the forest is used to avoid the generation of new nodes that increase this value while making all the elements splittable.

```

struct Tree {
    Integer      type;
    Integer      material;
    Integer      orientation[3];
    Integer      split_axes[3];
    Integer      refinement_levels[3];

    Integer      global_corner_numbers[Corners_per_element];
    Tree*        sons;
    Cut_info*    cut_information;

    Flag         mid_edge_point[Edges_per_element];
    Flag         mid_face_point[Faces_per_element];
};

```

Figure 9: Information stored for each tree node in C-like syntax

The array `global_corner_numbers` contains the global numbers of each element corner. `cut_information` tells if some boundary or material interface could not be fitted while generating the macro-elements. For each node that is not a leaf, `sons` points to node successors. While making all the elements splittable, the arrays `mid_edge_point` and `mid_face_point` keep track of the edge midpoints and face center points.

4.1.2 Storage of allocated Mesh Points in a Hash Table

Apart from the tree and the tables that contain the coordinates and the impurity concentration for each mesh point, Ω_{me} stores all allocated mesh points in a hash table. In order to identify each mesh point, integer coordinates are associated with each possible point location in the grid as follows: the corners are allocated integer coordinates 0 and $2^{N_{\text{max}}}$, respectively, N_{max} being the maximal admissible tree depth ($N_{\text{max}} = 20$ will fit most cases). The edge midpoint of $(0, 0, 0)$ and $(0, 0, 2^{N_{\text{max}}})$ has integer coordinates $(0, 0, 2^{N_{\text{max}}-1})$, and in general, the edge midpoint of $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$ has integer coordinates $((x_1 + x_2)/2, (y_1 + y_2)/2, (z_1 + z_2)/2)$. All these coordinates are guaranteed to be integer, since no edge gets split more than N_{max} times. The integer coordinates of the macro-element corners are multiples of $2^{N_{\text{max}}}$.

For each mesh point, the hash table stores the integer coordinates and the global point number. The size of the hash table is of the order of the number of mesh points; the number of nodes is of the same order, since the number of leaf nodes and the number of points are roughly equal. On the other hand, each entry of the hash table is significantly smaller than an entry of the tree; therefore, Ω_{me} does not require much more memory space per point than the octree version.

4.2 Generation of the Initial Grid fitting the Geometry

As we said before, in our first 3-D grid generator based on the conventional modified octree approach, elements along the boundary or interfaces were refined until they approximated the

geometry sufficiently according to an epsilon value. In this new implementation the geometry is fit as “exact” as possible at the beginning using the set of macro-elements described in Section 3.2. The next subsections describe a simple algorithm used and how far we can come using this concept.

4.2.1 Generation of the Initial *Macro-grid*

The geometrical description of structure includes the device boundaries and the material interfaces. The device is defined using a set of general polyhedra. Each polyhedron is described through a material value and a set of polygons, and each polygon through a set of points.

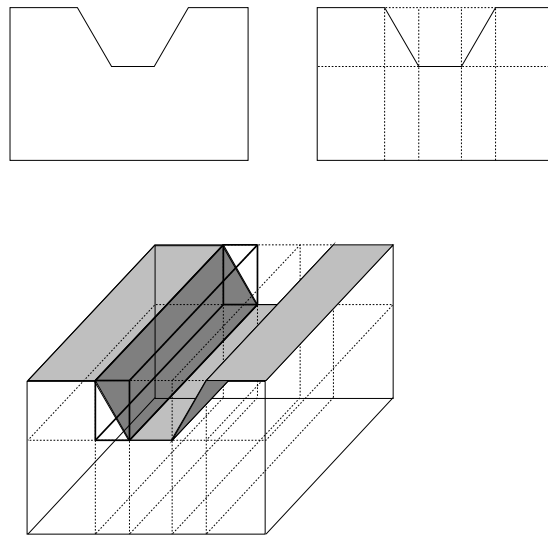


Figure 10: Example of a 2-D and 3-D tensor-product mesh

Using the point information, a 3-D tensor-product mesh is generated. A cuboid called the *bounding volume* is drawn which surrounds the perimeter of the device geometry with the minimum volume. Planes whose normal vectors are parallel to the x , y and z directions, respectively, are drawn through each point of the device geometry from one end of the bounding volume to the other (Fig. 10). The above part of the figure shows the 2-D case, where the lines correspond to the 3-D planes.

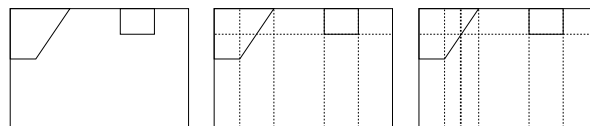


Figure 11: Sequence of steps during the tensor-product mesh generation

During the generation of a 2-D tensor-product grid, new points can appear because of intersections between the lines inserted and some boundary or material interfaces(Fig. 11). The center figure shows the tensor product mesh after a first step of the line drawing process, and the right one the final mesh. Lines have to be inserted until each point has its corresponding horizontal and vertical line. Extending this process to the generation of the 3-D tensor-product

grid, planes have to be inserted until each point is lying on three planes. There exists one case however, where this process reproduces always the same situation. New lines in 2-D (or planes in 3-D) are inserted as shown in Fig. 12. In the current implementation, this infinite loop is terminated according to a minimal size of the element edges and a slight modification of the device geometry.

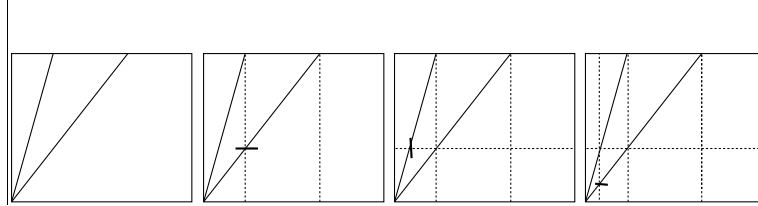


Figure 12: Problematic geometry

After the 3-D tensor-product mesh is generated, the bounding volume looks like a set of cuboids with some of them cut by the boundary or internal interfaces. Non-cut cuboids are already valid as macro-elements. Cut cuboids are tessellated into valid macro-elements using a set of predefined patterns. A predefined pattern consists of a set of macro-elements that fits a particular cut cuboid. Examples for simple cuboid cuts and their corresponding example, some simple cut cuboids and their tessellations are shown in Fig. 13. The left cuboid is tessellated into two rectangular prisms and the right one in three rectangular pyramids.

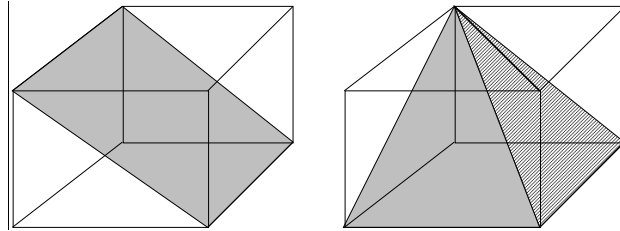


Figure 13: Patterns to fit cut cuboids

As final step, the material of each macro-element is set. Figure 14 shows the *macro-grid* for an oxide-isolated ECL bipolar transistor structure. In order to clearly identify the interface the two different material regions of the device have been “pulled apart” and the top has also been rotated (SiO_2 on the top and Si at the bottom). The zoomed view at the right permits to distinguish the type of some of the macro-elements used.

Ω_{me} also handles cut macro-elements if they satisfy Theorem 5. For example, the interface intersecting the left pyramid of Fig. 15 is transmitted through the sons in the refinement process as shown in the right pyramid of the same figure. Two cut rectangular pyramids and four rectangular prisms are generated. Finally, the cut pyramid is fitted using two tetrahedra. The Voronoi faces are the same as in a normal pyramid except for one of them. The vertical edge e has a Voronoi face belonging to two material regions.

4.2.2 How General is this Method?

Ω_{me} can fit any structure in an exact manner if the geometry can be partitioned into the set of

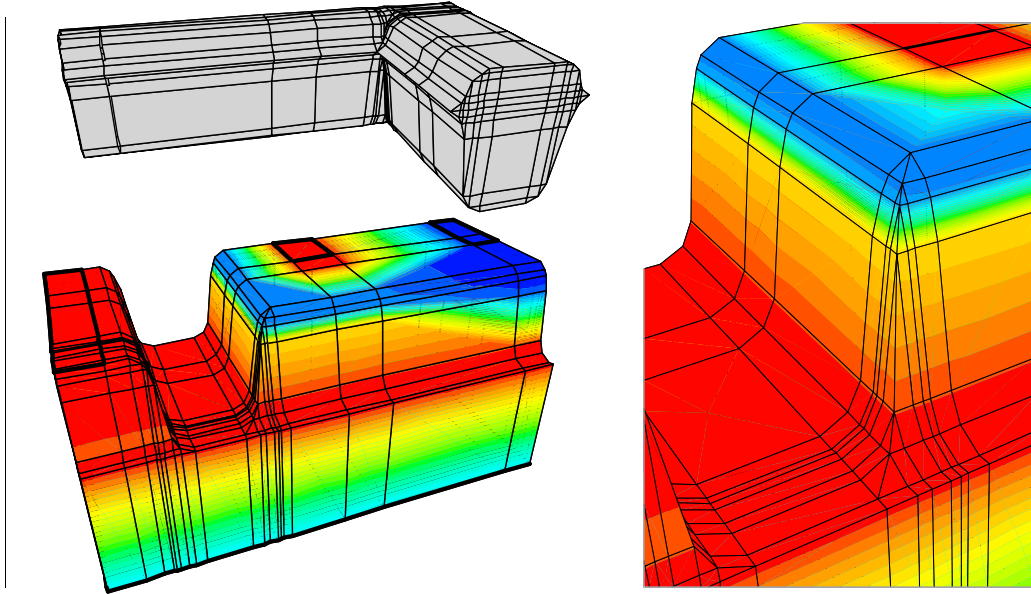


Figure 14: Macro-grid for the ECL bipolar transistor. The number of macro-elements is 2,069: 28 pyramids, 369 prisms and 1,672 cuboids.

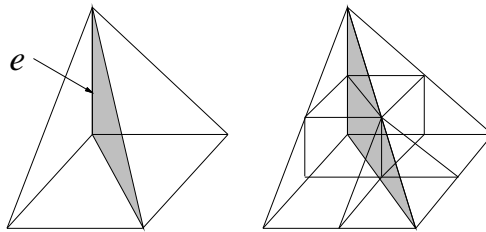


Figure 15: Transmitting cuts in the refinement process

macro-elements defined above. Intersected elements that cannot be tessellated using the set of macro elements described in Section 3.2 are recognized using predefined patterns and the grid generation process is aborted with a proper message.

The strategy presented here generates a complete partition after fitting the geometry with the advantage that elements can be always split at the edge midpoints. This considerably simplifies the element refinement and the tessellation of 1-irregular elements. The disadvantage is that grid lines needed to fit a particular geometry are propagated through the complete structure. Depending on the geometry, elements with very bad aspect ratios might be generated.

The current way to handle the infinite loops is still not satisfactory because of the large amount of time spent and the amount of unnecessarily elements generated. We are now working a better way to fit the device geometry.

4.3 Generation of an Irregular Grid Fulfilling Density Requirements

Once the initial domain has been split into macro elements, these are recursively refined until an adequate mesh density is obtained in all regions of the device. In Ω_{me} , it is assumed that for each element there exists an indicator telling across which axes the element needs further refinement.

It makes no difference whether the indicator is given a *priori* (for semiconductor devices, based on the variation of the impurity concentration within the element), or a *posteriori*, based on an estimate of the error in the solution. Elements considered too coarse are repeatedly refined until the final grid is appropriately dense in all regions of the device.

4.4 Generation of the Finite Element Mesh

The irregular grid is transformed into a proper finite element mesh. The basic idea is analogous to the algorithm used by Shephard and co-workers for their modified octrees: first, a graded tree is generated, and then all elements between coarse and refined device regions are subdivided into proper cells.

4.4.1 Generation of a 1-irregular Tree

Algorithm 2 makes an arbitrary mixed element tree 1-irregular. The complete tree is traversed, and leaves carrying quarter-edge or three-quarter-edge points are split further across those edges.

```

do
  nodes_refined = false;
  for (all leaves)

    /* Check which edges were split more than once, and store the
       axes parallel to these edges */
    for (all edges eij in the leave)
      if (mid_edge_point (pi, pj) in hash table)
        mep = mid_edge_point (pi, pj) ;
        if (mid_edge_point (pi, mep) in hash table or
            mid_edge_point (mep, pj) in hash table)
          store axis parallel to edge eij ;
        end if
      end if
    end for

    /* Refine node across all stored axes */
    if (any axes stored)
      Refine (current leave, stored axes);
      new_nodes_refined = true;
    end if
  end for
while (nodes refined);

```

Algorithm 2: How to make a mixed element tree 1-irregular.

Notice that when a leaf is split, this may add new quarter-edge points on leaves which have already been checked. Therefore, the traversal must be repeated until all leaves are taken care of.

In order to decide whether an element carries quarter-edge points, Ω_{me} checks whether each edge carries an edge midpoint, i.e., whether the edge midpoint is stored in the point hash table. If so, Ω_{me} checks whether any edge midpoint of the two halves of the edge is stored in the hash

table. Once all edges have been checked, Ω_{me} decides across which edges the node needs to be split.

4.4.2 Tessellation of 1-irregular Leaves

The irregular mesh satisfying the density requirements is made 1-irregular by traversing the mixed element tree as shown in Algorithm 3.

```

nodes_refined = false;
do
  make tree 1-irregular;
  nodes_refined = false;
  for (all leaves)
    Look up in hash table which edges are split and store it in pattern;
    if (pattern is not splittable)
      Insert proper points or refine leaf perpendicular to longest edge that
      not increase the deep of the tree;
      nodes_refined = true;
    end if
  end for
while (nodes_refined)

```

Algorithm 3: Make all irregular leaves splittable

The tessellation of 1-irregular elements is done template-based, as for Shephard’s original modified octrees. A 1-irregular element is represented by its split edges(pattern). The amount of different patterns depends on the shape of the element. A brick has twelve edges; then there exist 2^{12} different 1-irregular bricks. A prism has nine edges (2^9 different 1-irregular prims), and a pyramid has eight edges (2^8 different 1-irregular pyramids).

Several patterns are permutations of the same topology. They have been classified into equivalence classes or pattern types in order to manage the tessellation information. Each pattern type is associated with a tessellation and the required eccentricity condition(Definition 6).

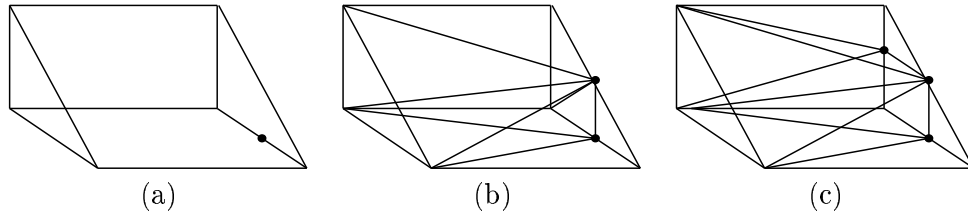


Figure 16: Inserting points in the refinement of a prismatic element. (a) Non-splittable 1-irregular prism. (b) Finite element tessellation inserting one point. (c) Finite element tessellation inserting two points.

Depending on the problem type, different criteria such as the insertion of key points and refinement across proper directions are applied to non-splittable 1-irregular leaves. For example, the 1-irregular prism leaf shown in Fig. 16 has a non-splittable topology according to Definition 6. One or two points can be inserted if the element has the required eccentricity. Otherwise, the prism is split in a way that the 1-irregularity condition is destroyed.

For elements with splittable topology which fail to fulfill eccentricity requirements both the insertion of points into proper edges and the refinement across the problematic direction are used. If there is no special indicator to determine the best direction, the longest edge that does not increase the depth of the tree is chosen in order to decrease the eccentricity. This is an acceptable strategy as eccentric 1-irregular elements are quite likely to need further refinement.

When a leaf is refined, the new points inserted may destroy the 1-irregularity of the tree or add a point on a edge of an irregular leaf which has already been checked. Therefore, at each iteration, the grid must be made 1-irregular and the main loop must be repeated until no more refinements are required during a complete traversal.

4.5 Leaf-wise Assembly of Voronoi Surfaces

Once all the 1-irregular leaves are splittable, each element must be tessellated into tetrahedra, pyramids, prisms and bricks and the corresponding cross sections are stored.

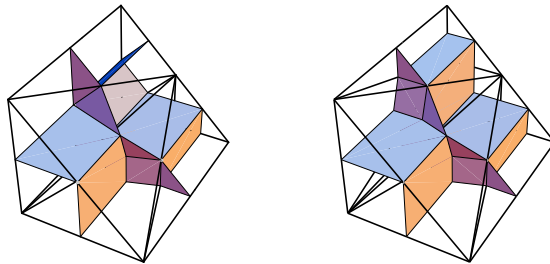


Figure 17: Perpendicular cross sections for two types of 1-irregular prisms

Each leaf is considered a grid *per se* and so the perpendicular cross sections are computed as if the boundary of the leaf was the boundary of the whole grid. Tetrahedra are allowed as a proper final element. The tessellation and the corresponding perpendicular cross section information is obtained from a hash table indexed by the local numbering of the split edges [13]. The cross sections are evaluated symbolically with *Mathematica*, a package for symbolic computation [15]. When Ω_{me} assembles the cross sections, it traverses all leaves once and adds the surface perpendicular to each edge to the total edge surface.

For illustration, Fig. 17 shows the finite element tessellation of two types of 1-irregular prisms together with the corresponding perpendicular cross sections for each edge of the tessellation.

5 Examples

We now present some representative examples to illustrate the current functionality of Ω_{me} . Figure 18 shows a zoomed view of a trench-isolated bipolar transistor as used in state-of-the-art high-speed ECL designs.

Figure 19 shows a short channel MOS transistor with surrounding LOCOS isolation. The whole device without grid is shown on the left side and a zoomed view on a part of the p-region under the gate is shown in the right.

Figure 20 shows a MOS-controlled thyristor with integrated MOS switches for turn on-off. The size of this structure is $10 \mu\text{m} \times 10 \mu\text{m} \times 500 \mu\text{m}$. The left view depicts the whole device and the right one is a zoomed view of the top part. The gate is a thin oxide layer of $0.02 \mu\text{m}$.

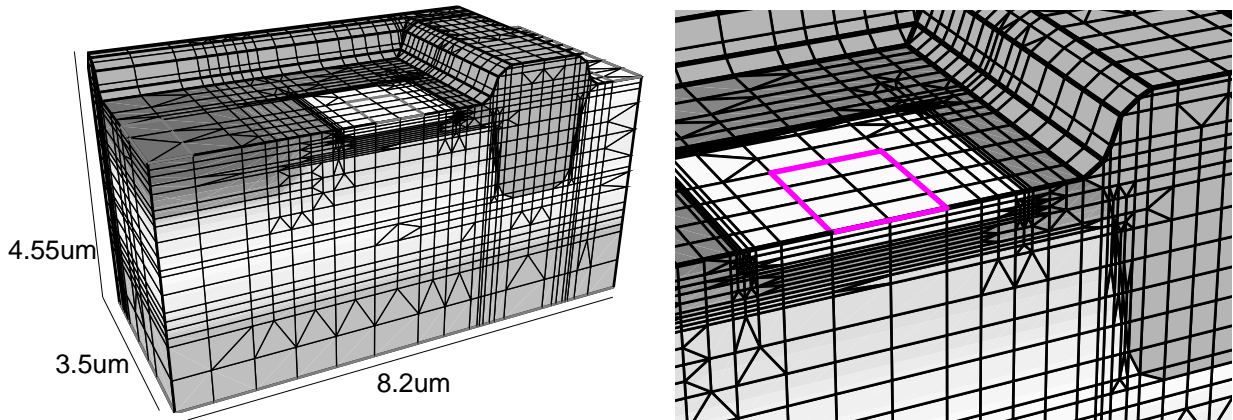


Figure 18: ECL bipolar transistor: A final grid at the left and a zoomed view at the emitter and p-channel at the right

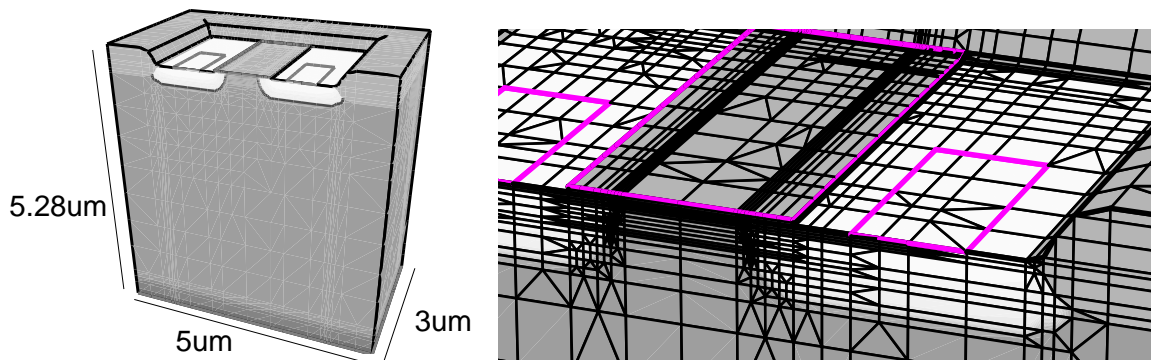


Figure 19: LOCOS: the whole device at the left and a zoomed view under the gate at the right

6 Performance

Table 1 shows run times, memory requirements and the mesh sizes for the examples presented in Section 5 and for a part of a n-well CMOS configuration (Fig. 21). All the examples were run on a SUN SPARCstation 1+ with 20Mb main memory.

The information given in the table helps to understand the empirical performance of the implemented algorithms. *#smooth. loops* indicates the number of tree traversals which must be performed to make all irregular leaves splittable once the required point density has been achieved. In Ω_{oct} , exactly one traversal is required, independent of the tessellated device, while in Ω_{me} the number of traversals depends on the device and on the required mesh density (cf. Section 4.4). *#add. points* refers to the points inserted during these smoothing loops. The percentage figures are relative to the mesh size before smoothing. Finally, *geometry* gives the time Ω_{me} spends generating the *macro-grid* for the devices with more complicated geometries.

Figure 21 shows four views of the CMOS structure: on the top the whole device together with a zoomed view of the mesh generated by Ω_{oct} and at the bottom, the same views for the mesh generated by Ω_{me} . Both grids were generated using the same input file. Therefore, they satisfy the density requirement specified by the user. Ω_{me} generates fewer elements than

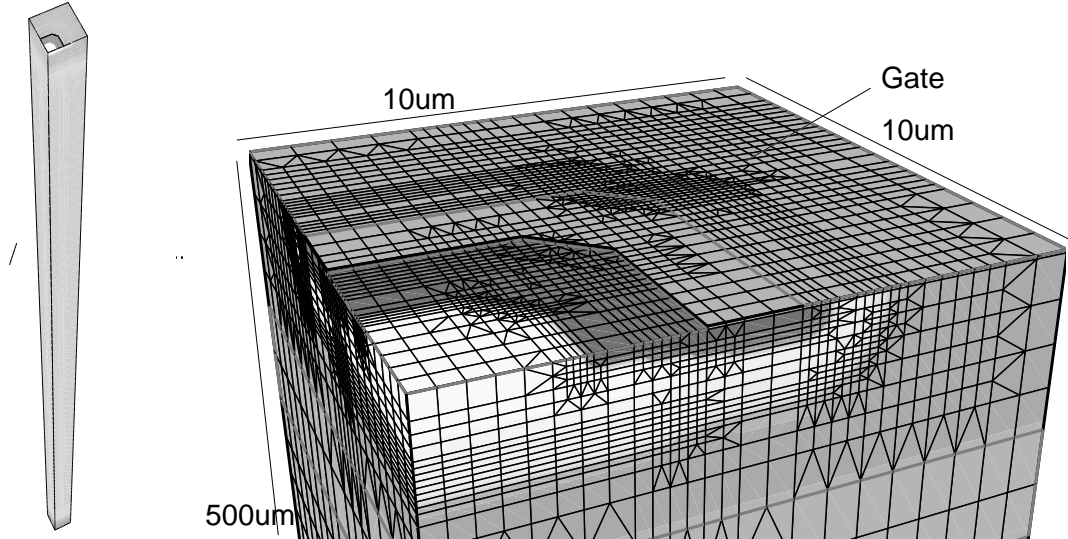


Figure 20: MOS-controlled thyristor: the geometry and doping profile at the left and a zoomed view on the top at the right

Device	CMOS		ECL	LOCOS	MCT
Ω Version	Ω_{oct}	Ω_{me}	Ω_{me}	Ω_{me}	Ω_{me}
# points	68820	13926	16729	16586	31846
# elements	100033	13008	20432	23409	41716
CPU [s]	284	165	825	554	860
memory (mb)	13,7	10,8	7,5	7,5	11,2
geometry [s]	-	-	546	198	41
# smooth. loops	1	3	16	22	15
# add. points	5891	57	5718	8296	5219
% add. points	9%	0.004%	51%	100%	19%

Table 1: CPU and memory requirements of Ω_{oct} and Ω_{me}

Ω_{oct} because it refines only across the doping gradient variation (the doping is represented by different gray tones); elements are subdivided isotropically in Ω_{oct} . The fact that both grids have the required mesh density does not say anything about the convergence of the PDE. It depends on how good the user can estimate these parameters.

The lack of neighbor retrieval functions in Ω_{me} leads to significantly higher run times per point compared to the original octree version. In particular, the algorithm which makes an arbitrary mixed element tree 1-irregular has a worst case complexity $O(N \log N)$, N being the number of leaves in the tree. This function must be called repeatedly to look for irregular leaves which do not satisfy the conditions of well-shaped elements. However, the significantly smaller meshes generated by Ω_{me} are certainly worth the degraded per point performance, leading to an overall gain.

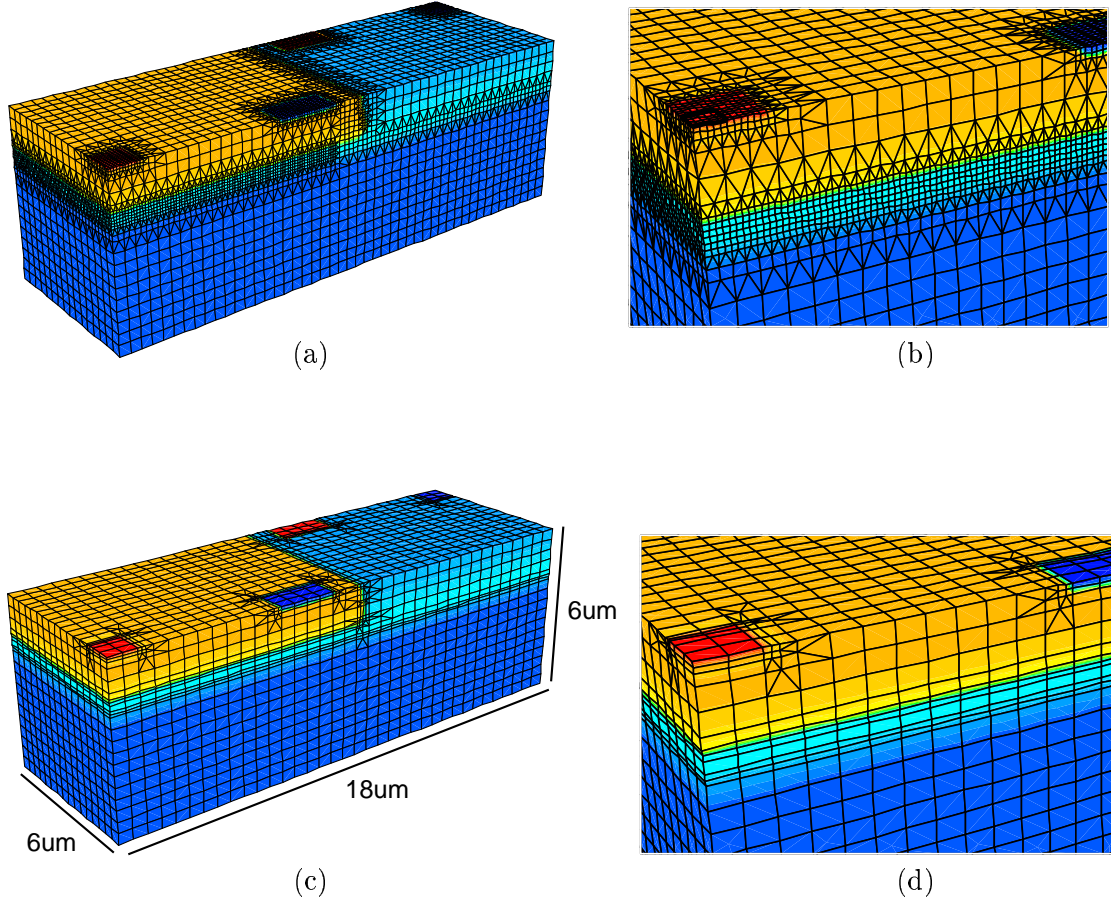


Figure 21: CMOS configuration: (a) shows the complete grid and (b) a zoomed view of the grid generated by Ω_{oct} and (c) and (d) the same views for the grid generated using Ω_{me} .

7 Conclusions

The grid generator Ω_{me} is used as front-end for the 3-D device simulators SECOND[16] and SIMUL[17] developed in our laboratories. It permits the construction of Delaunay meshes for 3-D semiconductor devices, fitting exactly the geometrical model specified by the user. Because the rectangular tetrahedron is not handled due to the reason explained in Section 3.2, Ω_{me} cannot fit completely general geometries. Ω_{me} uses rectangular tetrahedra only if a finite element mesh is required.

In comparison with the modified octree versions[2], Ω_{me} fits more complicated geometries with fewer 3-D elements. The main reason for this improvement is threefold. First, Ω_{me} uses different types of 3-D elements (rectangular pyramids, rectangular prisms and bricks) to fit the device geometry instead of using only cuboids. This avoids refinement necessary only to reach a reasonable approximation at the device boundary and interfaces. Second, elements are refined only along the necessary coordinate axes. This means, for example, that a cuboid can be partitioned into two, four or eight sons instead of only eight. Without these concepts, we

would not be able to generate meshes for very eccentric devices such as power devices (MCT in Fig. 20), because of the large number of points generated by a normal modified octree approach. Third, Ω_{me} handles very eccentric elements wherever possible. In looking for proper tessellations, element refined across its longest axis direction if proper points cannot be inserted. In order to guarantee convergence, no element is refined beyond the level of refinement attained before the 1-irregular step. Here all the refinement requirements are already fulfilled.

Currently, we are improving the performance and modifying the geometry fitting process. The new way to fit geometry should avoid the propagation of grid lines to the whole device and allow a better manipulation of complicated geometries. In addition, the time spent fitting the device geometry should be also reduced.

References

- [1] R. E. Bank, D. J. Rose, and W. Fichtner, "Numerical methods for semiconductor device simulation," *IEEE Trans. on El. Dev.*, vol. ED-30, no. 9, pp. 1031–1041, 1983.
- [2] P. Conti, N. Hitschfeld, and W. Fichtner, " Ω – an octree-based mixed element grid allocator for the simulation of complex 3d device structures," *IEEE Trans. on CAD/ICAS*, vol. 10, pp. 1231–1241, 1991.
- [3] A. Maus, "Delaunay triangulation and the convex hull of n points in expected linear time," *BIT*, vol. 24, pp. 151–163, 1984.
- [4] W. J. Schroeder and M. S. Shephard, "A combined octree/Delaunay method for fully automatic 3-d mesh generation," *Int. J. Numer. Methods Eng.*, vol. 29, pp. 37–55, 1990.
- [5] R. E. Bank, A. H. Sherman, and A. Weiser, "Refinement algorithms and data structures for regular mesh refinement," in *Scientific Computing* (R. Stepleman *et al.*, eds.), pp. 3–17, North-Holland, 1983.
- [6] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations* Users' Guide 6.0. Society for Industrial and Applied Mathematics, 1990.
- [7] M. A. Yerry and M. S. Shephard, "A modified quadtree approach to finite element mesh generation," *IEEE Computer Graphics and Applications*, pp. 39 – 46, 1983.
- [8] S. Müller, K. Kells, and W. Fichtner, "Automatic rectangle-based adaptive mesh generation without obtuse angles." *IEEE Trans. on CAD/ICAS*, 1992, in press.
- [9] P. Conti and W. Fichtner, "Automatic grid generation for 3d device simulation," in *Proc. of SISDEP 3 Conf.*, pp. 497–505, 1988.
- [10] M. A. Yerry and S. Shephard, "Automatic three-dimensional mesh generation by the modified-octree technique," *Int. J. Numer. Methods Eng.*, vol. 20, pp. 1965–1990, 1984.
- [11] M. S. Shephard, M. A. Yerry, and P. L. Baehmann, "Automatic mesh generation allowing for efficient a priori and a posteriori mesh refinement," *Computer Methods in Applied Mechanics and Engineering*, vol. 55, pp. 161–180, 1986.
- [12] E. K. Buratynski, "A fully automatic three-dimensional mesh generator for complex geometries," *Int. J. Numer. Methods Eng.*, vol. 30, pp. 931–952, 1990.
- [13] P. Conti, *Grid Generation for Three-dimensional Device Simulation*. PhD thesis, ETH Zürich, 1991. published by Hartung-Gorre Verlag, Konstanz, Germany.
- [14] B. Delaunay, "Sur la sphère vide," *Bull. Acad. Sci. USSR(VII)*, pp. 793–800, 1934.
- [15] S. Wolfram, *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, 1988.
- [16] G. Heiser, *Design and Implementation of a Three Dimensional, General Purpose Semiconductor Device Simulator*. PhD thesis, ETH Zürich, 1991. published by Hartung-Gorre Verlag, Konstanz, Germany.

- [17] S. Müller, N. Hitschfeld, C. Pommerell, K. Kells, M. Westermann, and W. Fichtner, “Technology-cad algorithms, implementations, and results,” in *Proc. NEC research symposium*, SIAM 1991.