High Performance Parallel Mesh Generation and Adaption

F. Stimpfl, R. Heinzl, P. Schwaha, and S. Selberherr

Institute for Microelectronics, TU Wien, Gußhausstraße 27-29, Vienna, Austria

Abstract. The continuing growth of complexity in physical models and the addition of more accurate geometrical features intensifies the weight placed on mesh generation. Driven by the increase of computational speed and the availability of multi-core CPUs current programming paradigms are not sufficient anymore to fully utilize the available computational power. A high performance mesh generation approach overcomes these difficulties by suitably combining multiple programming paradigms and following modern design guidelines. Parallelization and robustness of the algorithm are facilitated by employing a rigorous surface treatment, which not only enforces prescribed quality criteria such as the Delaunay property, but also allows to decouple the subsequent parallel meshing steps. We present a parallel advancing front algorithm capable of creating Delaunay conforming meshes.

1 Introduction

Modeling, generation, and adaptation of unstructured meshes is of utmost importance for scientific computing, especially in the area of Technology Computer-Aided Design (TCAD) [1]. Different fields of TCAD application impose a variety of different constraints and requirements on mesh generation, e.g., topography simulation requires a good approximation of surface elements, while ion implantation simulation requires a high mesh density near the surface, according to the gradient of the ion distribution. Diffusion simulations add a need for a fine mesh at interfaces in addition to a high mesh density near the surface. The complex field of device modeling even requires a completely different type of mesh, necessitating a remeshing step for the whole input structure. In summary, it can be observed that each simulation step has completely different requirements on the underlying spatial discretization. Therefore, meshing is still one of the major showstoppers in this field of scientific computing.

Meshing is the initial step for simulations and failing to properly control the meshing process will produce a mesh of bad quality and, therefore, can jeopardize or even completely prevent the chain of simulations. Changes made during the simulation process often bring the necessity to remesh or alter the structure during the course of the simulation. When utilizing complex input structures the remeshing steps can take an enormous amount of time, which delays all further processing steps and, therefore, slows down the simulation process as a whole. Both, quality of the mesh and remeshing steps, are issues which call out for robust high performance approaches.

The simulation of microelectronic devices such as transistors is an area of TCAD, which mostly makes use of finite volume schemes for discretization, due to their inherently flux preserving nature, which also implies the fundamental requirement of

a conforming Delaunay mesh. Most of the Delaunay mesh generation algorithms are based on Delaunay refinement, which always construct a convex hull and subsequently refine it. By following this approach additional difficulties on parallelization and high performance mesh generation are imposed.

Another issue which often occurs during simulation is the variation of element sizes of the input structures, e.g., diffusion simulation. On one side of the structure the mesh consists of very small elements, while the opposite side is made up of very large elements. The transition between these two sides results in difficulties for mesh generation and special mechanisms for handling this difficulties have to be applied.

A very important constraint in TCAD simulations is the possibility of scaling. Not only should the meshing approach work for small devices, but it should also be scalable for complex device structures meshed in parallel on multiple cores or on a high performance computing cluster, respectively. This constraint is supported by the current trend of increasing the amount of cores in a single processor.

High performance approaches therefore accelerate the necessity of parallelization techniques which are required to fully utilize the available computational power. This circumstance adds to the already complex mesh generation task, as geometrical and topological consistency has to be ensured, which requires particular attention in a parallel setting and necessitates the use of advanced programming techniques and paradigms to be implemented efficiently. The availability of robust high performance tools is therefore of utmost importance.

Traditional programming approaches are not sufficiently utilizing the increasing computing power even in desktop systems anymore. To tap this powerful and growing resource the application of modern programming paradigms is increasingly important for scientific computing. The concept of parallelization, which is often only applied reluctantly, as many of the already tested algorithms and implementations need to be rethought or rewritten, which of course entails new and thorough testing. Fortunately current compiler technologies already incorporate facilities fitting the multi-core nature of modern CPUs to support the development of parallel applications, e.g., the parallel STL which is part of GCC 4.2 [2] is accounted for and combined with already established partitioning tools such as METIS [3].

We present an approach to parallel meshing based on a combination of advancing front algorithms which optionally include the Delaunay property and thereby are able to yield suitable results for both finite elements and finite volume discretization schemes. Our approach first ensures that the input hull meets prescribed quality criteria before a volume mesh is generated. In case a Delaunay tessellation is requested, the conforming Delaunay property [4] is enforced by the surface treatment algorithm. It then proceeds with the generation of the mesh by using an advancing front algorithm specially adapted to consistently provide elements fulfilling the Delaunay property and avoiding colliding fronts. The main advantage of our approach is the ability to generate meshes using local feature size criteria, while being compatible to the upcoming multi-core processor designs by making use of state of the art programming techniques and paradigms.

2 Meshing Theory

A complex requirement of current Delaunay algorithms is the creation of a convex hull of the initial input from which the final mesh has to be extracted by recreating the given boundary of the initial structure. This issue may not only result in overhead, due to the construction of convex hull parts, which can be of substantial size and also have to be meshed just to be removed at the end of the mesh generation, but also due to numerical problems. This issue unnecessarily complicates and slows down the whole Delaunay mesh generation process.
The formal part given in the next section is derived and adapted from [4–6], which guarantees the consistency and the Delaunay conformity. The volume mesh generation is treated by an advancing front algorithm based on abstract rules [7] for the insertion of new elements during the advancing front algorithm. Throughout this paper the term tessellation is used as generalization of a triangulation in two dimensions or a tetrahedralization in three dimensions. Similarly the terms volume element and surface element are used to designate triangles and lines or tetrahedra and triangles in two or three dimensions, respectively.

Delaunay Tessellation
The definition of the Delaunay property is given first. The property was introduced by Борис Делоне (Boris Delone, Delaunay being the French transliteration) in 1934 [8] and can be generalized using the following empty n-ball claim.

Definition 1. *An n-ball is said to be empty, if it encloses no vertices of a set $V \subseteq \mathbb{R}^n$, where n is the dimension.*

Using this claim, a simplex, which consists of n vertices of V, is said to be Delaunay, if and only if there exists an empty n-ball that passes through these vertices.

Lemma 1. *Given a domain D containing the vertices V and the set of boundary elements B, then $\forall\, b \in B$ there is no vertex $v \in V$, which encroaches b, if b is Delaunay.*

Lemma 1 assures, that all boundary elements satisfy the Delaunay property and this lemma can further be extended to Theorem 1 to show the Delaunay property for the whole tessellation.

Theorem 1. *Let T be the set of volume elements of a tessellation of D. If $\forall\, t \in T$ is locally Delaunay then T is globally Delaunay.*

Proof. Consider a volume element $t \in T$ and a vertex $v \in V$ different from the vertices forming t. Due to the local Delaunay property v lies outside the n-ball of t. Because this is then true $\forall v$, the n-ball of t is empty, and because this is then true $\forall$ volume elements t, D is the Delaunay tessellation of V.

There exist two different concepts which extend the definition of the Delaunay triangulation for boundaries - the constrained Delaunay triangulation (CDT) and the conforming Delaunay triangulation. Both concepts have in common, that they start from an initial tessellation, which includes the convex hull, and refine the existing tessellation to fulfill the Delaunay property. When creating a CDT the boundary edges are preserved

and are not split into smaller edges by avoiding the insertion of additional vertices. An edge or triangle is said to be constrained Delaunay, if it satisfies the following two conditions. First, its vertices are visible to each other, meaning that no segment of the simulation domain lies between the vertices. Second, there exists a circle that passes through the vertices of the edge or triangle in question, and the circle contains no vertices of the triangulation which are visible from the interior of the edge or triangle [4].

In contrast to the CDT, where the boundary is not modified, when creating a conforming Delaunay tessellation the boundary is modified by inserting new vertices in order to satisfy the Delaunay property for all boundary elements. Both concepts aim to fulfill Lemma 1.

The next section gives an overview of the advancing front algorithm, which is explained using an example in two dimensions. The generalization to higher dimensions is possible.

Advancing Front Algorithm

For our Delaunay volume mesh generation, the advancing front algorithm is derived from the gift-wrapping algorithm, which can be specified n-dimensionally. It starts with a set of boundary elements. These boundary elements form the initial front which is advanced into the simulation domain. A boundary element of this set is chosen to form a new element, either with an existing point or a newly created point. The current edge is then removed from the front and the two new edges are, depending on their visibility, added to the front. This process terminates when no edges remain within the front.

The advantages of this method are the good control mechanism for the element sizes and the quality of the generated elements. A major drawback of this method is that the quality of the generated elements depends heavily on the quality of the boundary elements and the colliding fronts. Different implementations of this type of mesh generation technique suffer from severe robustness issues.

Due to the fact that the advancing front depends heavily on the quality of the boundary, we prepare the boundary according to the Delaunay properties defined in the previous section. Therefore, when starting from a Delaunay conforming boundary, the resulting advancing front will satisfy the Delaunay property only, if no additional points are inserted.

Our advancing front algorithm uses abstract rules [7] which define the procedure of mesh generation, e.g., how new points are inserted or how certain elements are treated during the meshing process. The rules are defined in a unit coordinate system and the current element is transformed to this unit coordinate system, a matching rule is applied, and the results are transformed back to the original mesh. The procedure of choosing a matching rule can be performed by various criteria, e.g., element size or element quality. The following will combine the meshing theory with the practical techniques.

3 Our Meshing Approach

The first step, the processing of the boundary, assures that all boundary elements conform to the Delaunay property according to Lemma 1. Not only the surface vertices but also the volume vertices are taken into account, when processing the surface to create a Delaunay tessellation.

Our proposed algorithm based on Lemma 1 is equal to the conforming Delaunay tessellation, but without the overhead of creating an initial tessellation first and without the overhead of cutting all elements between the boundaries out of the tessellation afterwards. An example for a processed boundary is given in Figure 1.

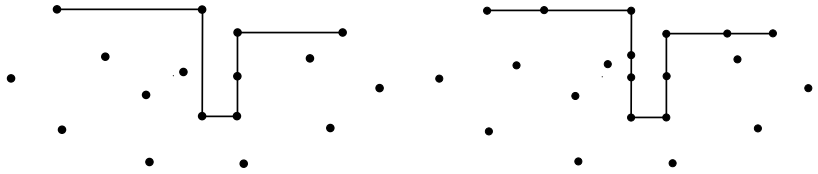

Fig. 1: An example of a conforming Delaunay triangulation. Before and after surface preprocessing step.

The refinement of a boundary element is performed, when a vertex in its vicinity exists, which would encroach this element and therefore violate Lemma 1. One straightforward method is to refine the boundary element by an orthogonal projection of the encroaching vertex onto the boundary element, as depicted in Figure 2. The created refined boundary element is split into new boundary elements, depending on the dimension of the boundary element, e.g., a projected vertex onto a boundary edge is split into two new boundary edges. This procedure creates new boundary elements, which satisfy Lemma 1 and, therefore, are locally Delaunay.

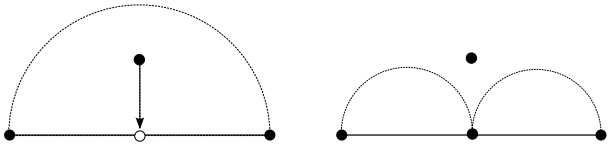

Fig. 2: A surface element and the circumcircle which is encroached by a volume vertex (left). The resulting two Delaunay surface edges, after the orthogonal projection of the encroaching vertex (right).

A second case exists, because the encroaching vertex is incident to another boundary element and, using an orthogonal projection, the created refinement would itself become an encroaching vertex, due to numerical inaccuracies. This situation may lead to an endless refinement loop, which limits the applicability of the orthogonal projection. For this case an azimuthal rotation of the encroaching vertex around the intersection of the boundary elements instead of the orthogonal projection is performed. An example for the azimuthal rotation is depicted in Figure 3. The result of this surface processing step is a conforming Delaunay surface tessellation.

The necessary projections and rotations to fulfill Lemma 1 are controlled by abstract rules as mentioned in the previous section.

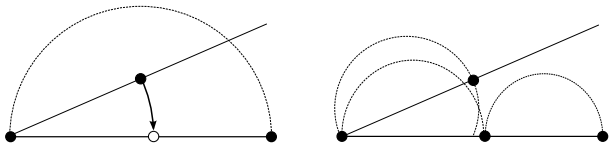

Fig. 3: An edge and the circumcircle which is encroached by a vertex on an incident edge (left). The resulting two Delaunay surface edges after the azimuthal rotation of the encroaching vertex (right).

In the subsequent step the advancing front algorithm traverses all existing boundary elements and creates new volume elements according to Lemma 1. The volume vertex closest to the boundary element, which does not encroach the boundary element, is used to create a new volume element [4].

Due to the fact that the chosen vertex is not encroaching, the resulting volume element satisfies the Delaunay property. Applying Theorem 1, if all elements are locally Delaunay, then the whole tessellation is Delaunay, which proofs, that the presented Delaunay meshing approach results in a Delaunay conforming volume mesh. Figure 4 depicts our developed parallel meshing approach, starting from the common surface treatment.

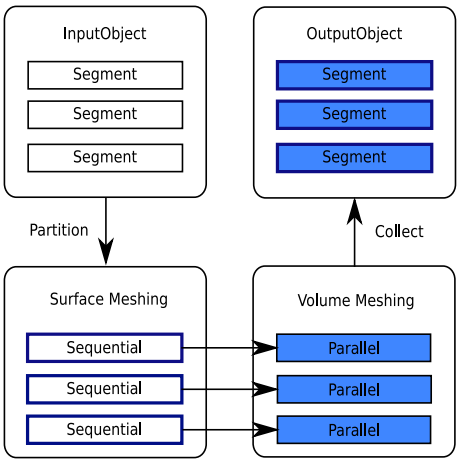

Fig. 4: An overview of the presented meshing approach. Starting from an initial input geometry the surface preprocessing step is done. The segments are meshed in parallel and in the final step the resulting meshed segments are merged into one output geometry.

4 Programming Paradigms

The implementation of algorithms related to advancing front mesh generation techniques is one of the most complex programming topic due to the combination of ge-

ometrical and topological issues. Geometrical robustness and accuracy problems can yield topological inconsistencies, whereas topological problems can severely circumvent the successful termination of the whole algorithm.

The matter of consistency is even more pronounced in a parallel environment, where consistency between the concurrent parts has to be accounted for explicitly.

To deal with these issues we have separated the geometrical and topological areas into different types of programming parts. Geometrical issues are treated by using generic programming and the outsourcing of this treatment into numerical libraries, e.g., interval arithmetic or exact numerical kernels like CGAL [9]. The precision of the used geometric predicates is essential to ensure that element consistency is maintained during the advancing front algorithm.

As outlined in the previous section, our approach yields a decoupled method which does not require communication between the parallel code parts. This makes the procedure appealing not only for parallelization using shared memory as provided, e.g, by OpenMP [10], but also for message passing interfaces such as Open MPI [11]. The current trend of deploying multi-core machines clearly favors the use of shared memory parallelization techniques, especially since they have begun to be integrated into the newest generation of the freely available compiler collection, GCC.

Automated parallelization can only be effective, if the compiler is supplied with sufficient semantic information as possible. This specification of algorithms at the required high semantic level is greatly facilitated by the use and combination of several programming paradigms, which at the moment is only efficiently supported in the C++ programming language [12]. The parallel STL is likely the first step in this direction, which emerging compilers are pursuing and is consequently picked up and used by our Generic Scientific Simulation Environment (GSSE) [13, 14] used for topological operations.

The importance of the use of several complementary programming paradigms becomes apparent, when considering how to best implement parallel tasks. In order to be reliable, parallel parts must not have side effects or explicit dependencies on global state information. While such a requirement needs to be specifically taken care of in procedural and object-oriented programming approaches, functional programming already inherently incorporates the required traits. However, functional programming has great difficulties when dealing with files, as these essentially represent frozen state information which cannot be accommodated in a purely functional setting.

The generic programming paradigm provides many features which have initially been envisioned for the object-oriented paradigm. However, since algorithms are usually woven into the data carrying objects, object-oriented development has problems reusing algorithms. The reusability of source code developed using the generic programming source code eases also debugging and maintenance.

The appropriate combination of several distinct programming paradigms can alleviate the shortcomings of the individual paradigms, while making the strengths available to the whole. The generic programming paradigm is well suited to procedurally deal with file and input/output operations by iterations, which can be used to supply information to functional code parts which are inherently parallel. Parallelization of the whole construct can then be achieved by simple partition of the iteration.

The following snippet of code shows a central part of the mesh generation application, using a GSSE domain, parametrized to a specific data type, as an interface for segments which are fed to the a functional meshing routine.

```
for_each(domain.segment_begin(),
         domain.segment_end(),
         generate_mesh(thread_id++));
```

The parallelization of the traversal of the segments of the domain by iterator partitioning is sufficient to parallelize the meshing procedure, due to the functional nature of the specification. It is therefore possible to develop and test algorithms in a sequential manner and then parallelize them by simple recompilation. This basic strategy remains the same, even for seemingly complex tasks.

However, a major caveat remains in this approach. The data types, to which the GSSE domain has been parametrized must not contain internal states, e.g., in the form of static member variables which prohibit parallelization.

The approach of combining several programming paradigms offers great flexibility for developing, testing, and quickly deploying new algorithms in a very efficient manner.

5 Examples and Benchmarks

The presented approach is demonstrated using examples from different fields of TCAD. It can be observed that the speed of the parallel approach reduces meshing time considerably, thus enabling the whole simulation process to quickly get a result, as shown in Table 1. Execution time can be decreased with increasing segment size and complexity.

Example	Sequential Meshing	Parallel Meshing	Num. points	Num. segments
Diffusion Example (Figure 5)	149 sec	59 sec	1.2e4	2
Levelset (Figure 6)	31sec	19sec	1.9e4	3
MOSFET (Figure 7)	74sec	46sec	3.6e4	7

Table 1: Comparisons of the mesh generation and included mesh adaptation times (in seconds) on AMD's X2 5600.

The following example shows device structures which have been meshed in parallel. The various segments are colorized differently to show the partition of the mesh.

6 Conclusion

The highly complex tasks of modeling, mesh generation, and adaption can greatly benefit from modern programming approaches and a multi-paradigm approach. The application of modern programming paradigms and implementation of a multi-paradigm development enables not only the incorporation of modern compiler technology, but also eases an orthogonal optimization approach.

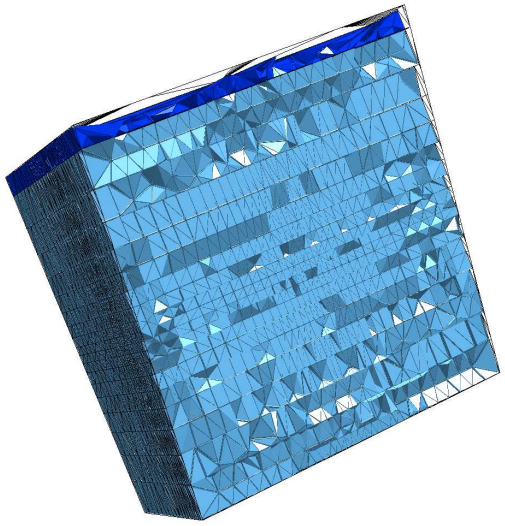

Fig. 5: TCAD process simulation, e.g., diffusion simulation requires an initial, spatially homogeneous and adapted distribution of a function space. Our volume mesh generation algorithm therefore incorporates a given point cloud to generate the illustrated mesh.

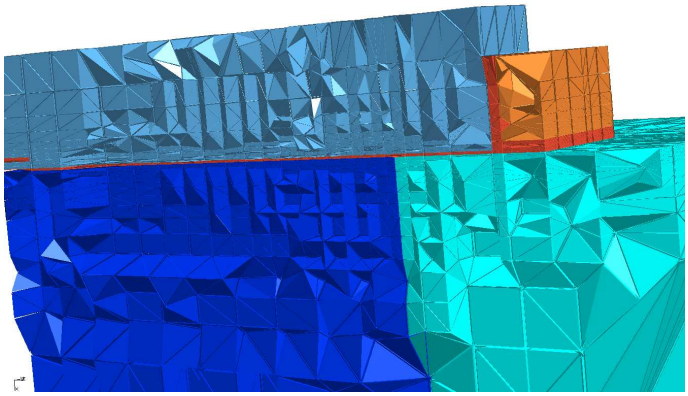

Fig. 6: Local feature size control enables meshing of thin layers of a three-dimensional device structure (marked in red) is made possible without imposing additional meshing overhead.

7 Acknowledgment

This work has been supported by the Intel Corporation and the Austrian Science Fund FWF, project P19532-N13.

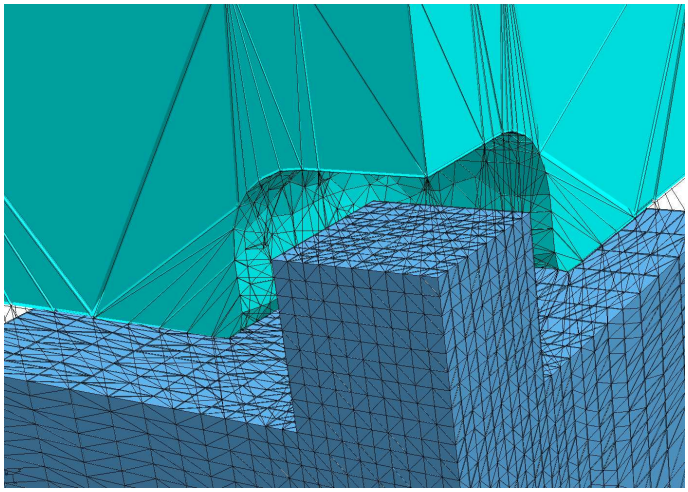

Fig. 7: Unstructured mesh representation of an extracted implicit surface used for moving surfaces in TCAD.

References

1. Heinzl, R.: Concepts for Scientific Computing. Dissertation, Technische Universität Wien, Austria (2007)
2. GNU: GNU Compiler Collection (GCC). http://gcc.gnu.org/.
3. Karypis Lab: METIS. http://glaros.dtc.umn.edu/gkhome/views/metis/.
4. Shewchuk, J.R.: Delaunay Refinement Mesh Generation. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA (1997)
5. Edelsbrunner, H.: Triangulations and Meshes in Computational Geometry. Acta Numerica (2000) 133–213
6. Ruppert, J.: A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. Journal of Algorithms **18**(3) (1995) 548–585
7. Schöberl, J.: NETGEN - An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules. Comput. Visual. Sci. **1** (1997) 41–52
8. Delaunay, B.: Sur la Sphère Vide. In: Izvestia Akademia Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, Moscow, Russia (1934) 793–800
9. Fabri, A.: CGAL - The Computational Geometry Algorithm Library. In: Proc. of the 10th Intl. Meshing Roundtable, CA, USA (2001) 137–142
10. Dagum, L., Menon, R.: OpenMP: An Industry-Standard API for Shared-Memory Programming. IEEE Computational Science & Engineering **5** (1998) 46–55
11. Graham, R.L., Shipman, G.M., Barrett, B.W., Castain, R.H., Bosilca, G., Lumsdaine, A.: Open MPI: A High-Performance, Heterogeneous MPI. In: Proc. of the 5th Intl. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, Barcelona, Spain (2006) 1–9
12. Garcia, R., Järvi, J., Lumsdaine, A., Siek, J., Willcock, J.: An Extended Comparative Study of Language Support for Generic Programming. J. of Functional Programming **17**(2) (2007) 145–205
13. Heinzl, R., Schwaha, P.: GSSE. (2007) http://www.gsse.at/.
14. Heinzl, R., Spevak, M., Schwaha, P., Selberherr, S.: A Generic Topology Library. In: Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf., Portland, OR, USA (2006) 85–93