Diss. ETH No. 14219

On boundary conforming anisotropic Delaunay meshes

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
Doctor of Technical Sciences

presented by
JENS KRAUSE
Dipl.-Phys.
Rheinische Friedrich-Wilhelms-Universität Bonn
born 26.11.1968
citizen of Germany

accepted on the recommendation of
Prof. Dr. Wolfgang Fichtner, examiner
Prof. Dr. Siegfried Selberherr, co-examiner

2001

Acknowledgement

First of all would like to thank Prof. Dr. Fichtner for giving me the opportunity to work and learn at the Integrated Systems Laboratory and Prof. Dr. Selberherr for reviewing my thesis.

Very important for my research work were my teachers Gilda Garretón, Norbert Strecker, and Luis Villablanca; without them I would have given up long long time ago. Irreplaceable Stefan Zelenka, with whom I could discuss the questions I dared not ask others.

I also want to thank the people I shared offices with over the years for the good atmosphere and fruitful side discussions: Nancy Hitschfeld, Paul Pfäffli, Andreas Pomp, Stefan Röllin, Markus Schaldach, Michael Schenkel, and Bernhard Schmithüsen. To this inspiring atmosphere contributed the entire institute, especially sectraterial, technical, and system administration staff.

This thesis was partially supported by the European projects PROMPT II (ESPRIT 24038) and MAGIC_FEAT (IST 1999-11433) and by the close collaboration with ISE AG in Zurich.

Contents

Abstract

Automatic mesh generation has become an integral part in solving partial differential equations numerically by the Finite Element Method or the Box Method as a variant of the Finite Element Method. Many engineering disciplines employ these methods to analyse new design concepts.

The framework of this work is the design of new semiconductor technologies. Generally, this is done in two steps: the modelling of the fabrication (process simulation) and of the electrical behaviour of the device (device simulation). The discretisation method of choice for the partial differential equations in this field is the Box Method, which calls for Delaunay meshes as a minimum requirement. On the other side, the boundary layer behaviour of the solution has to be modelled. A fine mesh with isotropic elements at material interfaces would contain too many points to be useful in practical applications. In order to save points, anisotropic elements are desired at these interfaces. In addition, the solution characteristics favours mesh lines interface parallel.

This work presents a novel method, normal offsetting, to generate the discretisation points in such a way that the induced Delaunay triangulation is anisotropic. The method uses a modified Advancing Front approach; in 2D layers of stretched quadrilaterals are added to the boundary and in 3D layers of flat prismatic elements are inserted into a surface mesh. Subsequently, a final triangular mesh is constructed.

This work focuses on a practical implementation of these ideas. In this respect, robustness of the generator to handle arbitrary geometries is very important. Especially in 3D, this stability is hard to achieve and

a constitutes major road block for a successful process simulation, which needs multiple re-meshing steps.

With the continuing miniaturisation of modern semiconductor devices 3D effects become more and more pronounced in the device characteristics. Also, the simulation of 3D effects becomes more important. This work, however, deals with 2D and 3D design and implementation of these new ideas.

Zusammenfassung

Die automatische Erzeugung von Simulationsgitter ist zu einem integralen Bestandteil bei numerischen Lösungsverfahren partieller Differentialgleichungen durch die Finite-Element-Methode oder der Box-Methode als eine Variante der Finite-Element-Methode geworden. Viele Ingenieurwissenschaften nutzen diese Methoden zur Analyse neuer Designkonzepte.

Diese Arbeit steht im Zusammenhang mit der Entwurf neuer Halbleitertechnologien. Im allgemeinen wird die Analyse in zwei Schritte aufgeteilt: die Untersuchung der Herstellung (Prozeßsimulation) und des elektrischen Verhaltens (Bauteilsimulation). Die Diskretisierungsmethode der Wahl für die partiellen Differentialgleichungen auf diesem Gebiet ist die Box-Methode, wodurch das Delaunaygitter als Minimalforderung begründet ist. Auf der anderen Seite muß das Grenzschichtverhalten der Lösung modelliert werden. Ein ausreichend feines Simulationsgitter mit isotropen Elementen an den Materialgrenzen enthielte zu viele Punkte, um für die Praxis brauchbar zu sein. Um Punkte einsparen zu können, sind anisotrope Elemente an solchen Grenzen von Vorteil. Außerdem ist wird die Lösung genauer wenn die Gitterlinien parallel zu den Grenzschichten konstruiert werden.

Diese Arbeit stellt eine neuartige Methode (normal offsetting) vor, die die Diskretisierungspunkte so anordnet, daß die Delaunay-Triangulierung dieser Punktmenge anisotrop ist. Diese Methode modifiziert den Advancing-Front Algorithmus: in 2D werden Schichten aus länglichen Vierecken der Geometriedefinition hinzugefügt und in 3D werden flache, den Prismen ähnliche, Elemente konstruiert. Schlußendlich wird ein Dreiecks- oder Tetraeder-Gitter erzeugt.

Die praktische Umsetzung dieser Ideen steht im Mittelpunkt der Arbeit. Deswegen ist Robustheit des Gittergenerators bei der Behandlung beliebiger Geometrien sehr wichtig. Besonders in 3D ist Stabilität ein Problem und stellt eines der bedeutenden Hindernisse auf dem Weg zur erfolgreichen Prozeßsimulation dar, die mehrere Gittergenerationen benötigt.

Die fortschreitende Miniaturisierung moderner Halbleiterbauteile bringt mehr 3D-Effekte im Schaltverhalten zum Vorschein. Daher wird auch die Simulation von 3D-Effekten notwendig. Diese Arbeit befaßt sich aber mit dem Entwurf und der Implementation der Ideen in 2D und in 3D.

Chapter 1

Introduction

This thesis deals with the development and implementation of algorithms for automatic mesh generation. Being part of a simulation environment the mesh generator has to conform with certain criteria that are being motivated in the second chapter. These criteria are linked with the main applications for which the meshes are used: in this work semiconductor process and device simulation are considered. While process simulation models the fabrication of semiconductor technology, device simulation analyses the electrical characteristics of the device. In both cases the so-called Box Method is used to discretise the partial differential equations (PDEs) that model the problems. It will be pointed out that the Delaunay triangulation is a necessary condition for the use in the Box Method.

Another requirement follows from the solution characteristics: the models in this field exhibit a strong boundary layer behaviour. For precise modelling of these boundary layers an interface conforming anisotropic mesh is required.

Further requirements stem from the integration of mesh generation into a simulation environment; it has to be reliable and not too demanding in the resources it uses.

To summarise the following criteria are demanded from the mesh generator, in the order of their priority:

- the generator has to work reliably,
- the mesh must comply with the Box Method Conforming Delaunay criterion,
- the mesh is anisotropic at certain interfaces,
- the elements of the mesh are of good quality,
- the mesh density is sufficient,
- the generator should be efficient and use modest computer system resources.

The second chapter features related work in mesh generation which is applied in this work or has inspired it partly. Namely, these are Delaunay methods, the Advancing Front Technique, the Octree method, and anisotropic meshing methods.

The main part of this dissertation discusses a method in 2D and in 3D to generate meshes that conform with the above criteria, which is called normal offsetting. As it turns out, the anisotropy of elements at a non-planar interface and the Box Method Conforming Delaunay criterion are partly in contradiction. It will be discussed that the achievable anisotropy is limited, but the limitations are independent of the method to construct the meshes.

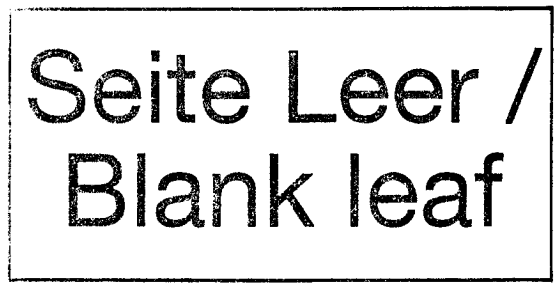
The third chapter is entirely devoted to the 2D implementation (*Noffset2d*); it uses a variant of the Advancing Front Technique to add layers of anisotropic quadrilaterals to the boundary. Another important feature is that the remaining polygonal void, which is not covered by the front, is triangulated directly.

These ideas can be partly transferred to 3D, which will be discussed in the forth chapter (*Noffset3d*). An Advancing-Front-like method adds layers of prisms to a triangular surface mesh. In order to triangulate the final void robustly, in 3D an indirect method is needed which uses a Delaunay technique.

The fifth chapter deals with the generation of surface meshes that are suitable for 3D normal offsetting. Near fold lines of the geometric model, it is important to generate a finer surface mesh with respect to

certain condition. With this refinement, the volume mesh created by normal offsetting can conform with the Delaunay condition.

The final chapter shows examples of meshes generated by this implementations. It covers 2D and 3D models and usability of these meshes is demonstrated in simulations.


Seite Leer /
Blank leaf

Chapter 2

Background and motivation

2.1 Triangulations and meshes

In the framework of this thesis, a mesh is a division of a polygonal domain Ω into primitives. As a particular case of a mesh, the triangular mesh is formed by triangles in 2D and tetrahedrons in 3D (as closed sets); other authors [Gar99] use mixed element meshes and allow also rectangles, prisms and bricks as elements. Let $\mathcal{T} = \{t_i\}$ be a set of those primitives, then they should cover the entire domain: $\bigcup_{t_i \in \mathcal{T}} = \Omega$. The mesh is conforming if the intersection $t_i \cap t_j$ for $i \neq j$ is either empty or a lower dimensional entity that is part of both elements, e.g. an edge that is shared by both elements.

The expression *triangulation* is used in 2D and in 3D, some authors, however, use in the latter case the words tetrahedralisation or tetrahedrisation. The difference between a triangulation and a triangular mesh is only minor. In general the term *mesh* is used for a triangulation that is fit for use in a simulation.

2.2 Process and device simulation

This section describes a brief introduction into the physical models for semiconductor process and device simulation A complete discussion is beyond the scope of this work and its full understanding is for the mesh generation problems not necessary. However, the drift-diffusion equations are listed, in order to discuss the discretisation scheme and to motivate the type of meshes needed.

In device simulation, a system of partial differential equations (PDEs) that couple electrostatic potential with the electron and hole densities is solved in the semiconductor material (e.g. references [MRS90, FRB83, Int99a]). Firstly, these quantities obey in the stationary case the following conservation laws, which are the Poisson equation and the continuity equations:

$$\begin{aligned} -\nabla \cdot (\epsilon \nabla \phi) &= q(N_p - N_n + N_D) \\ \nabla \cdot \mathbf{j}_n &= qR \\ -\nabla \cdot \mathbf{j}_p &= qR \end{aligned}$$

$N_{n/p}$	:	electron/hole density
$\mathbf{j}_{n/p}$	:	electron/hole current density
ϕ	:	electrostatic potential
R	:	pair generation/recombination rate
N_D	:	net doping concentration
q	:	elementary charge
ϵ	:	material dependent electric permittivity.

The concentration N_D reflects the electrical activation of the semiconductor by doping it with donor and acceptor impurities. The rate R includes advanced models for electron-hole pair generation and recombination.

The current densities in the drift-diffusion model are driven by a diffusive part (proportional to ∇N_x) and convective part which is proportional to the electric field:

$$\begin{aligned} \mathbf{j}_n &= \mu_n(-qN_n\nabla\phi + k_BT\nabla N_n) \\ \mathbf{j}_p &= \mu_p(-qN_p\nabla\phi - k_BT\nabla N_p) \end{aligned}$$

$\mu_{n/p}$: electron/hole mobility
k_B : Boltzmann's constant
T : temperature.

The boundary conditions (in terms of variables ϕ, N_n, N_p) that are applied to the problem are of different type, depending on the character of the interface. At Ohmic contacts, the values for potential and carrier concentrations are prescribed, whereas at artificial interfaces the normal fluxes (e.g. $\mathbf{j}_n \cdot \nu = 0$) of these quantities are set to zero [MRS90].

Apart from device simulation, a second field of application for this mesh generator is process simulation.

The modelled processes are [Int99b]:

Ion Implantation For ion implantation two different approaches are used. Firstly, analytical models which use combinations of Gaussian, Pearson, and exponential functions are used to approximate the dopant distribution after implantation. The Monte Carlo approach computes a database of particle trajectories and averages the final distribution. These models take the full layer system and the orientation of the critallographic axis with respect to the ion beam into account.

Diffusion The equations for dopant and oxidant diffusion are similar to those used in devices simulation, i.e. they contain diffusive and convective currents. Modern models do not only describe the dopant concentration, but also couple them with point defects such as interstitials, vacancies, and clusters of different species. Also the chemical reaction rate in presence of an oxidising atmosphere is taken into account.

Oxidation In the case of oxidation the geometric changes due to volume expansion of the oxide and material consumption are modelled, which results in a movement of the boundary. Viscoelastic models [PZSF00] compute a stress pattern that couples with the dopant and oxidant diffusion.

Deposition, etching, lithography These process steps can often be modelled with purely geometric models. The rates for deposition and etching can be localised and realistic simulations are possible. Again the boundary movement makes the mesh generation a challenging task.

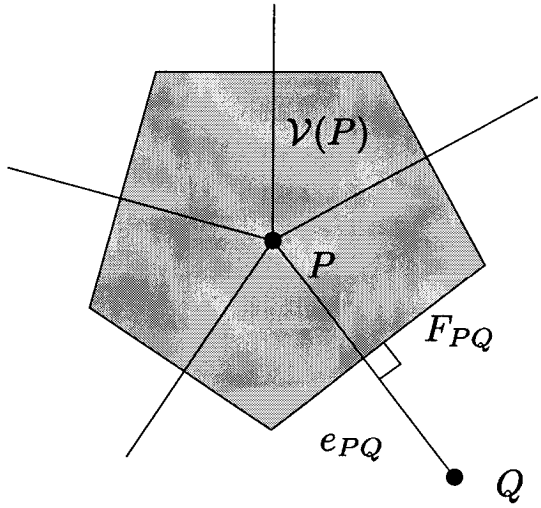


Figure 2.1: *Voronoï-cell $\mathcal{V}(P)$ of point P in 2D. Two cells $\mathcal{V}(P)$ and $\mathcal{V}(Q)$ are separated by edge mid perpendicular lines (planes in 3D)*

These fabrication steps introduce changes in the geometry or the dopant distribution and the mesh becomes invalid or not well adapted to the current state. These changes can be accounted for in some cases by local adjustments of the mesh, but the generation of a complete new mesh cannot always be avoided. In general several of these re-meshing steps are necessary in order to simulate an entire process flow. In this context, the mesh generation must be robust and automatic.

2.3 Box Method and the Voronoï-diagram

The discretisation method commonly used in this field is the Box Method (BM, e.g [Mül94, BRF83]). The primary idea of the discretisation is to associate each node P with a control volume $\mathcal{V}(P)$. The integrals over $\mathcal{V}(P)$ of the conservation laws in the form $\nabla \cdot \mathbf{j} = S$ are studied. Applying Gauss' theorem gives:

$$\int_{\partial \mathcal{V}(P)} d\mathbf{F} \cdot \mathbf{j} = \int_{\mathcal{V}(P)} dV S.$$

The choice for control volumes is discussed in [Gar99]; it turns out that the *Voronoï-cells* are the appropriate boxes (Fig. 2.1):

Definition 2.1 *Given a finite set of points* $\mathcal{S} \subset \mathbb{R}^d$. *For each* $P \in S$ *the* Voronoï-cell *is defined as:*

$$\mathcal{V}(P) = \{x \in \mathbb{R}^d \quad | \quad \forall Q \in \mathcal{S} \, \|x - P\| \leq \|x - Q\|\}.$$

The Voronoï-diagram $\mathcal{V}(\mathcal{S})$ *is the collection of all Voronoï-cells of points in* S.

In other words, the interior of $\mathcal{V}(P)$ contains the points in space that are closer to P than to other points of $\mathcal{S}$. In that sense the *Voronoï-diagram* represents a partition of space. The outline or a *Voronoï-cell* is always a convex polygon, which is not necessarily closed; the points on the boundary can have cells that reach into infinity because the general definition refers only to point sets that is not limited by a boundary.

In order to discretise equations some approximations have to be applied: the source term S is assumed to be constant in $\mathcal{V}(P)$ (mass lumping) and the current density through the face F_{PQ} (see Fig. 2.1) that joins cells $\mathcal{V}(P)$ and $\mathcal{V}(Q)$ is estimated by a constant $j(P,Q)$:

$$\sum_{Q,\exists \text{edge}(P,Q)} F_{PQ}\, j(P,Q) = \text{vol}(\mathcal{V}(P))S(P). \qquad (*)$$

The Scharfetter-Gummel Box Method uses the Bernoulli-function $B(t) = \frac{t}{e^t - 1}$ to approximate the current [SG69]:

$$\begin{aligned} j_n(P,Q) &= -\frac{\mu_n kT}{e_{QP}}[N_n(Q)B(\triangle_{PQ}) - N_n(P)B(-\triangle_{PQ})] \\ \triangle_{PQ} &= \frac{q(\phi(Q) - \phi(P))}{kT} \end{aligned}$$

For small arguments, i.e. small differences in the electrostatic potential, the Bernoulli-function is $B(t) \approx 1 - \frac{t}{2}$ and the standard finite difference scheme is restored. Originally, the result is only valid in 1D, but it is also applied in higher dimensions. The Scharfetter-Gummel approximation is the basis for the numerical stability of this discretisation method [Mül94].

The standard finite difference scheme leads to oscillations in the solution if the drift current dominates the diffusion current [Mül94]. Because

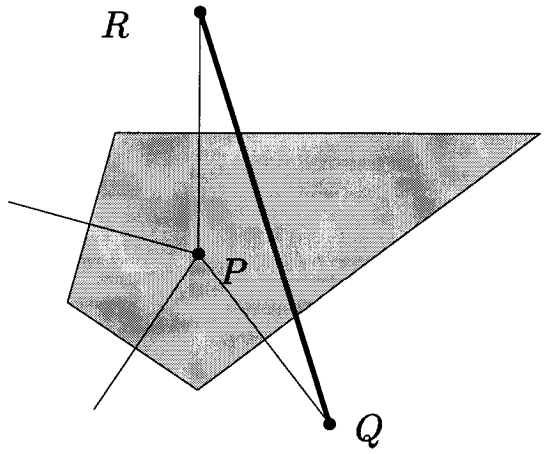


Figure 2.2: *At a material interface, the BM does not allow Voronoï-cells of point volume points P not cross material boundaries like RQ, because material dependent properties cannot be modelled properly.*

of its stability the Scharfetter-Gummel scheme cannot be replaced in the field of device simulation.

The equations (∗) together with the current and source approximations and the appropriate boundary values form the system of discretised equations for the carrier concentrations and the potential. The system is non-linear if the current and source terms depend non-linearly on the concentrations and the potential. In such a case an iterative Newton-solver is employed.

If material interfaces are taken into account, a stronger condition has to be imposed. In order to model material dependent properties correctly, the Voronoï-cells of volume nodes must be closed, i.e. they do not cross interfaces like shown in Fig. 2.2.

2.4 The Voronoï-diagram and the Delaunay triangulation

For the following it is important to note that the Voronoï-diagram is the (graph-theoretical) dual [For92] of the Delaunay triangulation (DT), which is defined later. This duality means that two nodes are connected by an edge in the DT if the mid-perpendicular line of this edge appears

in the Voronoï-diagram. Therefore, the DT is a necessary condition on the meshes for the BM. In general the dual of the Voronoï-diagram is not composed solely of triangles or tetrahedrons. But the higher order elements are convex and an arbitrary triangulation of these elements forms a DT.

In the following the *Delaunay triangulation* is defined and some other expressions that go with it.

Definition 2.2 *Given a point set $\mathcal{S} \subset \mathbb{R}^d$, a simplex (triangle or tetrahedron) composed of points from $\mathcal{S}$ is a* Delaunay-simplex *iff the interior(!) of the circumsphere*[1] *does not contain any points of $\mathcal{S}$.*

Definition 2.3 *A triangulation $\mathcal{T}$ of a point set $\mathcal{S} \subset \mathbb{R}^d$ is a* Delaunay-triangulation (DT) *iff all simplexes of $\mathcal{T}$ are* Delaunay-simplexes.

From the duality with the Voronoï-diagram the theorem follows directly:

Theorem 2.1 *Given a set of points then there exists a DT.*

The DT is not unique, however. The non-uniqueness is always related to cospherical points. These points form polyhedrons that can be triangulated in different ways into *Delaunay-simplexes*. For the Box Method all these triangulations are equivalent because they induce the same Voronoï-diagram.

In what has been discussed so far, the triangulation does not match with a specific geometry description. In fact, boundary faces and edges are not necessarily part of the DT. A DT that matches a given geometry is a called a *conforming*-DT.

Another important term in this respect is the *Delaunay-face* (*Delaunay-edge*). It opens the way to local algorithms:

Definition 2.4 *Given a triangulation $\mathcal{T}$. A face (an edge) f with neighbours t_1 and t_2 is called a* Delaunay-face (Delaunay-edge) *iff the following condition is met. The point p_1 of t_1 which is opposite to f is not inside the circumsphere of t_2, and vice versa.*

[1]Here, and in the following 'sphere' is used a generic term, it means *circle* in 2D.

By virtue of the following theorem [For92], a local test is sufficient to prove whether a triangulation is Delaunay.

Theorem 2.2 *Given a triangulation $\mathcal{T}$. The statements are equivalent:*

- *$\mathcal{T}$ is a DT.*
- *All faces(edges) in $\mathcal{T}$ are* Delaunay-faces *(*Delaunay-edges*).*

The requirement of closed Voronoï-cells is translated in the formulation of triangulations, by a stronger condition on boundary edges and faces. This leads to the definition of the *Box Method Conforming Delaunay Triangulation (BMCDT)*:

Definition 2.5 *A Triangulation is called a* Box Method Conforming Delaunay Triangulation (BMCDT) *if it matches a given input geometry and the interior of (minimal) circumspheres of the following items are point-free:*

- *tetrahedra (3D), triangles (2D),*
- *triangles (3D), edges (2D) at material interfaces or surfaces, and*
- *edges at material interface that join non-planar interface or surface triangles (3D only).*

In some cases a *constraint* DT is a useful temporary structure. It uses the definition 2.4 and the notion of a constraint; constrained edges and faces are desired in the triangulation, e.g. material boundaries can be considered as constraints.

Definition 2.6 *Given a triangulation $\mathcal{T}$ and a set of constraint $\mathcal{C}$, the triangulation is a* constraint *DT with respect to $\mathcal{C}$, iff*

- *all edges or faces of $\mathcal{C}$ are present in $\mathcal{T}$ and*
- *all edges or faces of $\mathcal{T} \backslash \mathcal{C}$ are* Delaunay faces *or* Delaunay edges.

To be useful, the set of constraints should be conforming, e.g. if edges intersect each other, they do it at a common end point. In that case, the *constraint* DT exists in 2D but unfortunately not in 3D.

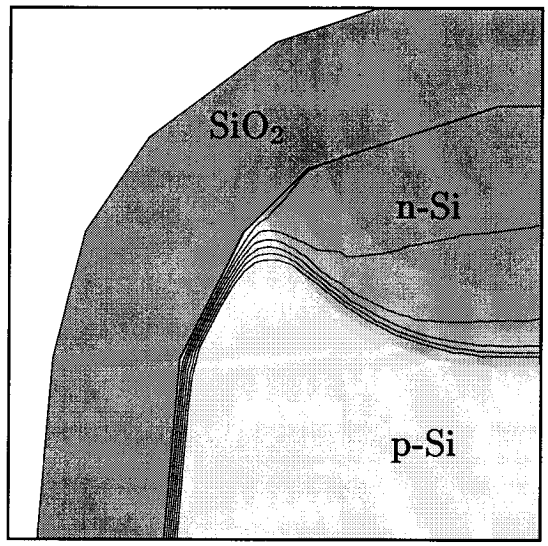


Figure 2.3: *Current confinement at the* Si/SiO_2 *interface in an IGBT (Insulated Gate Bipolar Transistor). The lines of constant current density are shown.*

2.5 Boundary layers and anisotropy

Another important mesh requirement stems from the solution characteristics of diffusion-convection-type equations, which exhibit boundary layers (e.g. in Fig. 2.3). Those layers have to be resolved by the mesh.

A striking example for a boundary layer is the current confinement in a thin channel in MOSFET[2] devices [KSVF00a]. Here, the discretised current density on interface edges parallel to the current is very large compared to the current densities on other edges. Due to the fact that the edge current densities are not projections of the current density on the elements the other edges do not carry the same current. If the current-parallel edges now have vanishing Voronoï surfaces the dominant current density terms do not contribute to the total current. Only for very fine meshes of this type the other edges carry sufficiently large current densities to approximate the interface current density.

Consequently, a desirable mesh contains edges parallel and orthogonal to the local current densities, which cannot be fulfilled for both carrier types simultaneously but approximately for the sum of both contributions or the dominant part.

[2]Metal Oxide Silicon - Field Effect Transistor

In *a priori* mesh generation the current density is generally not known to the mesh generator. In the situation described above, however, it is known that the current flows underneath the Si/SiO_2 interface, i.e. a geometric feature known in advance. The normal offsetting addresses this part of the mesh generation problem.

Other areas where anisotropy is advantageous are

- p-n–junctions,
- diffusion fronts in process simulation,
- material interfaces in process simulation to model segregation effects, and
- moving interface in mechanical models of process simulation.

2.6 Methods of automatic mesh generation

In the following, a few algorithms for mesh generation are cited from literature. These methods are either utilised directly or with modifications or are otherwise important to this work.

Delaunay connector

As stated before, a DT exists for a given point set; there are several methods to construct this DT:

- The incremental insertion methods [Bow81, Wat81] start with a triangulated box enclosing all points. The points are inserted one by one (Alg. 2.1), by removing a star-shaped cavity, which is refilled using the new point as the tip of a series of tetrahedra. The cavity is chosen in such a way that the resulting triangulation is *Delaunay.*
- Sweep, divide-and-conquer [PS85], and modified advancing front [Fle99] methods try to sort points spatially, so that only elements

Algorihm 2.1: *Incremental Delaunay construction*

INPUT: $\mathcal{T}_i = DT(\mathcal{S})$ in R^d, and a point P	
	Find simplices $t \in \mathcal{T}$ with P inside circumsphere of t; $t \rightarrow C(P)$.
	Remove $t \in C(P)$ from $\mathcal{T}$ (forms a void).
	Let $\partial C(P)$ the outline of $C(P)$, i.e. a list of edges/triangles
	Triangulate the void with triangles/tets with face $r \in \partial C(P)$ as base and P as tip
OUTPUT: $\mathcal{T}_{i+1} = DT(\mathcal{S} \cup \{P\})$	

are constructed that are part of the final DT. This can be advantageous in terms of time complexity, but all points must be known in advance.

- In 2D an arbitrary triangulations can be transformed into a DT by successive edge-flips [Law72].
- The generalisation of a transformation-based approach into 3D can get stuck in local minima. Villablanca [Vil00] introduces an algorithm to continue in those situations and conjectures that this procedure terminates for convex domains.

To be robust these methods have to make sure that the geometric tests (e.g. the in-sphere-test) work reliably. In the incremental approach, for instance, the cavity must be star-shaped.

These methods do not create a suitable mesh, because they only connect a given point set to form a 'balanced' triangulation. The point locations are found by separate algorithms.

Boundary enforcement

The boundary of the geometric model is not necessarily respected by the DT. The missing edges and faces have to be recovered by local transformations (edge and face swaps) or point insertions.

In 3D point insertions are inevitable. The methods use heuristic methods without proof of termination [She97b], if they try to minimise

the number of points needed to recover triangles.

Different methods are known to place points to recover boundary faces:

- internal points [GHS91]
- boundary points [She97b]
- *a priori* point creation on the boundary [Péb98].

Delaunay refinement

The term *Delaunay refinement* is used for techniques to find locations of refinement points and how to insert them into the mesh. In such generators a triangulation of the boundary points is created first and new points are inserted iteratively into the volume. After each point insertion, the mesh is optimised locally with respect to the Delaunay criterion, e.g. by using the incremental Delaunay construction.

The refinement points are usually inserted at the following locations:

- centres of circumspheres (i.e. Voronoï-centre) [HS88]
- edge-mid point [BG97].

The elements that violate some shape or size quality criteria are candidates for refinement. For some methods, mathematical proofs [She97b, Rup95, HR00] exist that certain quality bounds are met, often with impractical assumptions on input geometry like lower bounds on angles.

Advancing Front Technique (AFT)

The *Advancing Front Technique* [LP88, Löh96, MH95] creates points and their connectivity at the same time. In 3D, starting from a surface mesh tetrahedra are added to fill the void of the region to be meshed. A list of triangles is maintained as the current front. At each step one triangle from this list is designated as a base of a new element. In search of

an additional point to form the element the following candidates are considered: new points at different locations, existing points at connected elements, and existing points at opposite parts of the front. The decision is drawn using quality criteria; this is the strength of this method, because element quality and size can be controlled locally. Also, the validity of a new element must be checked, in particular that the new element does not intersect other parts of the front. This cannot be done by local tests, so data structures for spatial search are employed.

The AFT finishes when the front is empty. In 2D this is always possible, but in 3D situations are possible where no tetrahedron can be added to a base triangle. In that cases deletion of elements can open the necessary space to continue the construction. However, there is no prove of convergence and implementations suffer from closure problems [Sev97, Sch97].

The methods by Marcum and Weatherill [MW95] and by Frey et al. [FBG96] are of interest to this work. These references combine the AFT with Delaunay methods. The mesh vertices are created in the manner of AFT but they are inserted into a triangulation by a Delaunay algorithm. In that way the closure problem in 3D vanishes. The validity test which is a global search in the classical AFT becomes a neighbourhood search in the reconnection method.

Quadtree/Octree

The Octree method is very popular in the field of device simulation [Gar99, Fle99, Int99c][3]. It starts with the bounding box of the model as root cell and successively splits cells in 4 (quadtree in 2D) or 8 (octree in 3D) children. Cells are refined to resolve geometry or data function or, to comply with mesh refinement criteria given by the user. Also, to achieve a smooth transition between a coarse and a fine mesh the difference in tree levels between adjacent leaves is limited to one of two.

In the final step the mesh is extracted from the tree structure and the material associations are reestablished. Herein lies the difficulty of

[3]This implementation shares the input languages for geometry descriptions and simulation fields with a commercial mesh generator (MESH-ISE), which uses the Octree method. Throughout this text links to this generator are indicated in footnotes. The methodology, however, is not influenced by this link.

the method, because templates are used to match the configuration in one cell. Since only a limited number of templates can be implemented, further refinement splits may become necessary to find a simpler configuration. This increases not only the number of points but it is not even guaranteed to terminate. Especially, situations where two material interfaces cross one cell are hard to handle.

This method can easily create anisotropic elements by splitting cells in one coordinate axis only, but therefore anisotropy is restricted to coordinate axis. The Octree method is very fast and stable if the model contain only axis aligned faces. But for arbitrarily oriented faces it needs many points and is unstable in recovering material associations.

In 2D the quadtree method is the only approach that is able to generate obtuse angle free meshes [BE92].

Anisotropic methods

The *anisotropy* of an element measures how much an equilateral element needs to be deformed to attain this shape:

Definition 2.7 *Let l be the longest edge of a triangle or a tetrahedron and ρ the radius of its inscribed circle or sphere. The* anisotropy *or* aspect ratio *is the ratio:* $A = \frac{l}{\rho}$.

According to this definition an equilateral triangle has the *anisotropy* of $\frac{\sqrt{3}}{6}$, which is the minimum.

The scalar value for the *anisotropy* by itself is not very helpful: the orientation of its longest edge needs to be considered.

The literature, e.g. in the review article by George and Hecht [GH99], usually expresses the desired edge length for anisotropic meshes by a metric field g. This function is defined on the domain $\Omega \subset \mathbb{R}^d$ (which is assumed to be closed and simply connected):

$$g : \Omega \to \mathbb{R}^{d \times d}$$

where the matrix is symmetric and positive definite in Ω. Given a differentiable path $\gamma : [0, 1] \to \Omega$ connecting to points $P = \gamma(0)$ and $Q = \gamma(1)$

the length of this path with respect to the metric field is defined as:

$$d_g(\gamma) = \int_0^1 dt \sqrt{\gamma'(t) \cdot g(\gamma(t))\gamma'(t)}.$$

In a Riemannian space, *the* distance between P and Q is the length of the geodesic, i.e. the infimum of d_g for all γ connecting P and Q. This is computationally too expensive for practical mesh generators, and in many cases the (Euclidean) straight line is sufficient:

$$d_g(P, Q) = d_g(t \to P + t(Q - P)).$$

A mesh conforms with this metric if for all its edges holds $d_g(P, Q) \approx 1$. In practise, algorithms try to achieve $d_g(P, Q) < 1$, i.e. edges much shorter than unity are tolerated, but longer ones are not.

An implementation of an anisotropic Delaunay kernel has to determine the circumsphere of an element, e.g. by finding its circumcentre, which is the point that has the same distance to all points in the element. It is tedious to solve this problem with the straight line approximation, which contains an integration if g varies. George et al. [GB98] propose to evaluate the metric only in one point and measure the distance by

$$d_{g(X)}(P, Q) = \sqrt{(Q - P) \cdot g(X)(Q - P)}.$$

They show if the insertion point is taken as the sampling point that the cavity in the Delaunay kernel is star-shaped.

The usual isotropic mesh generation scheme is a special case of this procedure by setting

$$g = \frac{1}{h^2} id,$$

with the identity matrix id and the desired edge length h.

Such a method using an *anisotropic Delaunay connector* creates from a given point set a mesh with an anisotropic connectivity. This is problematic with the Box Method, because it relies on a Delaunay mesh in the Euclidean sense, which is in contradiction. Therefore it is important to create an *anisotropic point set* but to use a Euclidean Delaunay-connector.

The objective of this work is to create an interface conforming mesh; so the interface can be used as support to create an anisotropic point

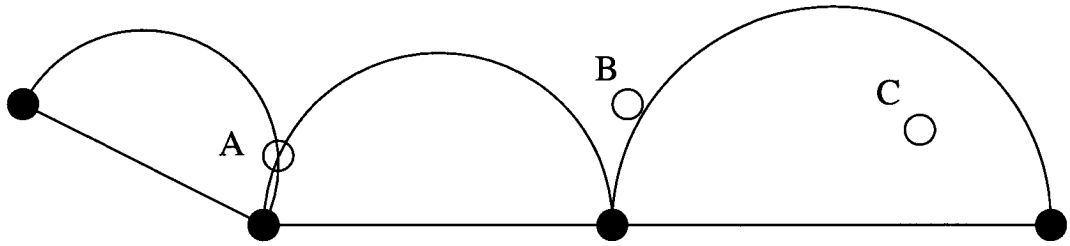


Figure 2.4: *Only points A and B can be used to form a mesh that is at the same time Delaunay and anisotropic, C intrudes the diametral sphere of an interface edge.*

set. The locations of the points must be chosen in such a way that the edges form an anisotropic mesh, if a Euclidean Delaunay-connector is applied. Figure 2.4 illustrates this: points that are images of interface points in local normal direction can be close to the interface without intruding the spheres of interface triangles.

Other references use similar approaches. In 2D Johnston and Sullivan [JS92] add quadrilaterals to the interfaces. In 3D references [PK96, KKM95] add layers of prismatic elements to interfaces. All references do not cope with the problem to create strictly Delaunay meshes, because their application do not use the Box Method.

2.7 Quality, optimisation and adaptation

The numerical correctness of a simulation depends on certain quality criteria on the grid. These criteria come in two flavours, as *shape* conditions and as *size* conditions.

The *shape* conditions try to measure the quality of an element irrespective of its size. The properties that are compared are the in-radii, circumradii, edge lengths, areas, angles and dihedral angles [Fle99].

A very popular measure is the ratio of the longest edge and the in-radius of a triangle of a tetrahedron: $Q^t = \frac{l_{max}}{\rho}$. A quality mesh would minimise Q^t for each element or the sum over all elements for a mesh.

For some types of equations that are discretised using a Finite Element method (FEM) there exist proofs that this is the appropriate

measure. Unfortunately such proofs are not known for the Box Method. However, experience shows that large angles and highly connected node must be avoided [HR00].

Some mesh generators focus on avoiding small angles to create quality meshes. However, when anisotropic meshes are to be created, then small angles are part of desired elements: e.g. in the 2D normal offsetting stretched quadrilaterals are cut into two triangle having small angles. Therefore in this work more focus is laid on avoiding *large* angles.

On the other hand, the *size* quality measures the density of mesh nodes, like the edge length. The desired mesh density is, for a first simulation, defined by the user, or – in an adaptive approach – derived from error estimates on a previous simulation [KSVF00a].

2.8 Aspects of computational geometry

Complexity

The choice of algorithm is often guided by optimising its complexity in space (use of memory resources) and time (number of operations). The $\mathcal{O}$-notation is used to express the complexity: $\mathcal{O}(f(n))$ means that for large problems of size n the algorithm needs a constant times $f(n)$ units of resources.

In general, the complexity of a mesh generator cannot be expressed in such a way, because it contains a collection of algorithms with different complexities. Deterministic algorithms like the Octree are an exceptions here, because the running time mainly depends on the number of cells in the tree. But for iterative refinement algorithms, where point insertions are governed by local quality criteria, predictions for the running time are not possible. Also, the construction of a coarse mesh can be more difficult than that of a fine mesh, and thus take longer.

Other than complexity, practical consideration are also taken into account for an optimal implementation: robustness, easy implementation, and expected characteristic of the input (e.g. sophisticated search algorithms are not needed for only a few items).

Exactness of predicates and algorithms

Mesh generation has to fight with a common problem in computational geometry: the imprecision of computations due to round-off errors. This can lead to incorrect solutions of algorithms; wrong results in predicates can cause infinite loops or exceptional situations when the algorithm cannot continue. The problem is related to the fact that only a limited number of digits (*bits*) can be stored on a computer but an unlimited or an infinite number of digits is needed to express the final and temporary results.

The situation is slightly different for predicates, i.e. a function $\mathbb{R}^m \rightarrow \{true, false\}$, if they use only additions, subtractions, and multiplications. In such a case, algorithms can internally use higher precision to store temporary results with exact values. The procedure then can return the exact answer, often by a simple sign check. This introduces some overhead, but fast algorithms are available that are output-sensitive and only extend the representation if it is necessary [She97a].

In algorithms which compute new locations of points divisions, square roots, and transcendental functions cannot be avoided, and rounding of the results becomes necessary. This can lead to conflicting situations, e.g. an algorithm that computes the intersection point of two lines rounds its output so it fits into the number representation. An exact predicate finds in the general case that the computed point does not lie on the lines!

The use of tolerances can help in these situations: the predicates then return an *undecidable* boolean value, if a certain floating point number falls within a tolerance interval around zero. These intervals can be calculated *a priori* [KW98] or dynamically depending on the input to the predicate [Vil00]. The calling algorithm would decide on the strategy to follow. In the above example the predicates testing whether the intersection point lies *left-of* and *right-of* the lines evaluate both *undecidable* which can be interpreted that the point lies on the line.

A particular algorithm can sometimes find alternative ways to handle degenerate cases. For example an AFT generator has to check, on which side of the base face a candidate point lies. In a critical case the tetrahedron that is formed has such a bad quality that it must be rejected also from these considerations.

2.9 Modularity

The different tasks needed to complete the generation of a mesh can be implemented in modules. This makes it possible to replace individual algorithms by other modules. In this work however, they are grouped around the normal offsetting, and the proposed implementations in this work are tuned for this purpose. Table 2.2 displays the various modules in the sequence of their invocation. The 2D and 3D implementations differ in some respect so they are shown separately. Also, a 2.5D generator is mentioned which handles the surfaces meshes for the 3D generator. In some cases two possible paths have been studied: one method using a standard AFT with intersection test and an alternative method using point insertion with local reconnection. The latter method has some advantages especially in 3D, as it is discussed in Sec. 4.4.

2D	
2D-ISOLINE(25) 2D-THIN(36)	
2D-REFINE-CURVATURE(39) 2D-BOUNDARY(35)	
2D-NOFFSET(26) 2D-TRIANGULATION(31) 2D-DELAUNAY(32)	2D-TRIANGULATION(31) 2D-DELAUNAY(32) 2D-RECONNECT(33)
2D-REFINE(31)	

3D	
3D-ISOSURFACE(70)(25D-PARAMETRIC-MAP(67)) "3D-BOUNDARY" ⇒ 2.5D	
3D-NOFFSET(42) 3D-INCREMENTAL(48) 3D-EXTRACTION(50)	3D-INCREMENTAL(48) 3D-RECONNECT(53)
3D-REFINE(55) 3D-BMCDT(52)	

2.5D
25D-TRIANGULATION(60) 25D-RECONNECT(67)
25D-REFINE(61)

Table 2.2: *Modules for 2D, 3D and, 2.5D generator. The numbers in parenthesis refer to the page in this thesis.*

Chapter 3

2D case: Noffset2d

The 2D version of normal offsetting is discussed as a modified Advancing Front Technique. The node creation, the global intersection test, and local operators are introduced as well as boundary meshing algorithms that are adapted to normal offsetting are described. Finally, an alternative approach that works without a global intersection test is proposed.

3.1 Input to the algorithm

Normal offsetting in 2D takes as input a polygonal boundary description where each material region is described by one or many simple single-connected polygons. A polygon is said to be simple if there is no pair of non-consecutive edges sharing a point. This boundary description is known as *planar straight line graph* (PSLG[BE92]). For multi-domain models this means in particular that at the interface between two regions, the polygons for both regions must use the same points.

In the framework of adaptive meshing it can be useful to include isolines of certain data functions defined on the domain [2D-ISOLINE].

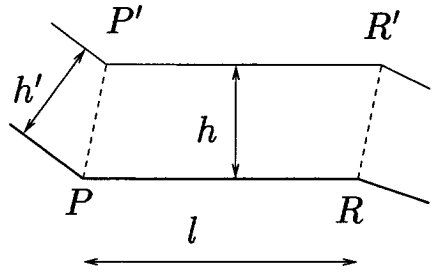


Figure 3.1: *The building block of normal offsetting: a quadrilateral.*

Examples are diffusion fronts in process simulation and p-n–junctions in device simulation. In this chapter it is assumed that these lines have been computed by some means and have been inserted into the boundary description. In that manner isolines are treated like material interfaces.

Apart from this geometric input, the algorithm also needs refinement information, which are:

- the height h and the length l of the anisotropic elements at the boundary (Fig. 3.1),
- a coarsening factor f by which the anisotropy decreases for elements further away from the interface,
- maximum number of layers, and
- a maximum edge length that is allowed for any edge in a particular material region.

A detail description on how to specify these data is postponed to appendix B; in particular these parameters can be localised.

3.2 A 2D string algorithm [2D-NOFFSET]

The normal offsetting approach aims at creating interface parallel and orthogonal mesh lines. This is done by creating layers of quadrilaterals (Fig. 3.1) with thickness h, which is done for each material region separately and only one layer at a time. The starting front of this process is

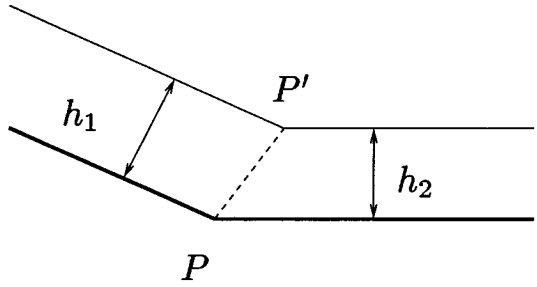


Figure 3.2: *Construction of an image point*

created from the input by segmenting the edges. When a quadrilateral is created the base edge is removed from the front and the parallel edge is inserted into this list. The lateral edges (like PP' in Fig. 3.1) are not part of the front, but they separate the meshed area from the unmeshed void. When a neighbouring quadrilateral is added, the lateral edges lose their role as separator; but this is not always the case.

This procedure has similarities with Advancing Front Technique (AFT) type mesh generators. In contrast, quadrilaterals are created instead of triangles and not all edges are a candidates as base for a new element.

3.2.1 Node creation

Each node P of the current front generates one image point P' (Fig. /ref-fig:pointlocation2d). The main goal is that the edges of the new front are parallel to those of the old front edge. Given the normal vectors at the two front edges attached to P: $\mathbf{n}_1, \mathbf{n}_2$; the parallel lines can be expressed by

$$\begin{aligned} (x - P) \cdot \mathbf{n}_1 &= h_1 \\ (x - P) \cdot \mathbf{n}_2 &= h_2. \end{aligned}$$

The shift vector $\mathbf{s}_{||}$ to the image is then:

$$\kappa = \det(\mathbf{n}_1, \mathbf{n}_2) = \mathbf{n}_1 R_{90}(\mathbf{n}_2) = -\mathbf{n}_2 R_{90}(\mathbf{n}_1)$$

$$\mathbf{s}_{||} = \frac{1}{\kappa}(h_1 R_{90}(\mathbf{n}_2) - h_2 R_{90}(\mathbf{n}_1)),$$

where R_{90} is the rotation matrix (by $\pi/2$): $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$.

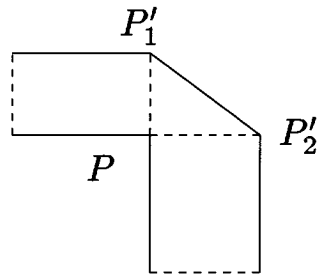


Figure 3.3: *Multiple images at interior angles larger than* $3\pi/2$.

If the normals are (almost) parallel this computation is not numerically stable, and there is no solution if h_1 and h_2 differ. In those cases the following average is the best compromise:

$$\mathbf{s}_{av} = \frac{1}{2}(h_1\mathbf{n}_1 + h_2\mathbf{n}_2)$$

In order to achieve a continuous transition between the two methods a weighted average is used with the weights:

$$\begin{aligned} w_{||} &= \kappa^2(h_1 + h_2)^2/\mathbf{s}_{||}^2 \\ w_{av} &= (1-\kappa^2)(h_1 - h_2)^2/\mathbf{s}_{av}^2 \\ \tilde{\mathbf{s}} &= \frac{w_{||}\mathbf{s}_{||} + w_{av}\mathbf{s}_{av}}{w_{||} + w_{av}} \end{aligned}$$

As the front advances it happens that frontal nodes have only one front edge attached; the image point is then found in the direction of the normal of this edge.

There is one exception to the *one image* rule: if the interior angle exceeds $3\pi/2$ two images are created in the first layer in the direction of their normals (Fig. 3.3).

3.2.2 Front advancement

As the front advances into the unmeshed area several algorithms are employed to ensure validity and quality of the mesh.

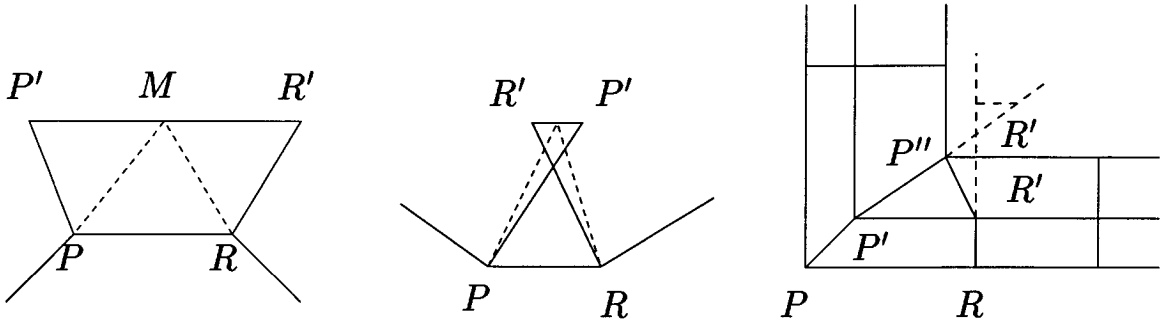


Figure 3.4: *Local quality tests; left: long edge, middle: twisted element, right: twisted element that would happen in the consecutive layer.*

Local tests After the image points of the node of one layer are created, the quadrilaterals that would be created are checked for quality. These algorithms are local in nature (Fig. 3.4):

- long edges (compared with the maximum edge length in this material region),
- twisted elements, and
- some twisted elements can be detected before they occur and can be avoided.

The pictures in Fig. 3.4 also show the amendments to the front: the quadrilaterals are replaced by triangles or pentagons, that are divided into three triangles. If these repair algorithms fail the front is stopped locally at this edge and the quadrilateral is removed.

Global tests Besides these local tests a global intersection test is necessary. Collisions of the front with other parts of the front must be detected because the front is stopped where such intersections occur. The search must be thorough, because an over-looked intersection lets the front run over other elements; in that case the mesh is invalid and the generator runs into an infinite loop. On the other hand, such a global intersection test is time consuming. To speedup this operation, a candidate edge for creation is checked with the exact intersection test only against those edges that have an overlapping bounding box. The search for overlapping bounding boxes is performed in an Alternating Digital Tree [BP91]. This binary tree splits axis aligned cells in two halves and

alternates at each level the coordinate axis of subdivision. Since bounding boxes in 2D are defined by four values the four dimensional version of the tree is used.

The time complexity for this search is – with these considerations – $\mathcal{O}(n \log n)$.

Coarsening The mesh is already coarsened because the thickness of each layer increases as the front advances. Another algorithm tries to increase the length of the front edges as the front progresses, so that a smooth transition to a coarse and isotropic volume mesh is possible. As shown in Fig. 3.5 the line at node P is not continued and the two quadrilaterals are merged to one pentagon. The criteria to this are:

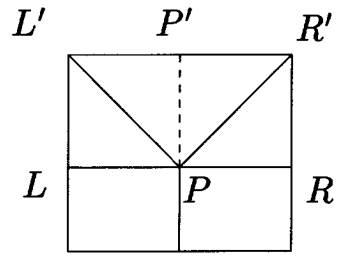


Figure 3.5: *Front coarsening: a mesh line is terminated at P*

- The angle $L'PR'$ has to be smaller than $\frac{\pi}{2}$.
- The angle LPR has to be larger than $\frac{3}{4}\pi$.
- The edge $L'R'$ has to be shorter than the maximum edge length for this region.

The idea here is that the merged situation is locally Delaunay so that P' is not reintroduced later.

Figure 3.6: *Left: bad quality elements are generated in the transition where the normal offsetting thickness jumps drastically; right: a better quality can be achieved by subdividing quadrilaterals*

Anisotropy enhancement As mentioned before the marching distance h can vary along the front, but at flat parts of the front an abrupt variation leads to elements of bad quality which are generated to fill the gap (Fig. 3.6 left). A better way to achieve higher and lower anisotropic elements close to each other is to, to create layers with smaller anisotropic quadrilaterals and in a second to step cut some elements parallel to the front (Fig. 3.6 right). By this subdivision the anisotropy can be enhanced by powers or two. In the transition region neighbouring elements are only subdivided with a difference of one level, thus avoiding highly connected nodes.

3.3 Extraction of the final mesh [2D-TRIANGULATION]

Normal offsetting finishes if no space is left to add further elements or if the maximum allowed number of layers is reached. Since normal offsetting does not fill the void a polygon remains, which is triangulated by a divide-and-conquer approach: the polygon is cut at reflex points until convex polygons or triangles are created [PS85].

At a reflex point the polygon has an obtuse interior angle. A polygon having no reflex points is convex and can be easily triangulated (by connecting an arbitrary point with all others). Thus, the number of reflex points measures the difficulty to triangulate the polygon. By cutting at reflex points the two child polygons have fewer reflex points and the problem becomes successively simpler. This simple algorithm has proven sufficient for the applications of this mesh generator.

3.4 Isotropic refinement and Delaunisation [2D-REFINE]

Normal offsetting does not in all cases create a sufficiently refined volume mesh, so further refinement is necessary. Also the quality (wrt. large angle and high connectivity, which are considered harmful for the Box Method [HR00, Vil00]) is improved by the following refinement

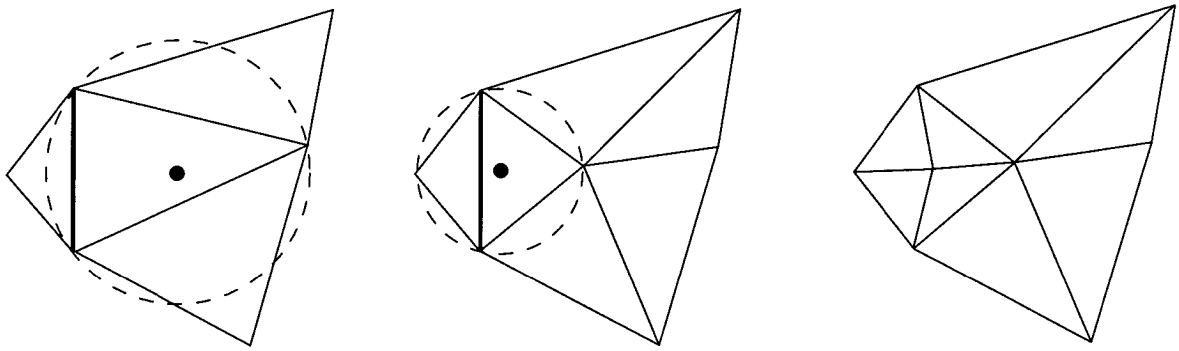


Figure 3.7: *In order to destroy an edge, more than one refinement point at neighbouring triangles might be necessary.*

algorithm.

The *isotropic* refinement algorithm relies on a BMCDT, which is constructed form the triangulation by successive edge flips [2D-DELAUNAY] (see Ref. [Law72, Vil00]). Non-Delaunay interface and boundary edges are refined by perpendicular inter-sectors, if these points are not to close to endpoints of the edge (in which case the edge is refined at the perpendicular inter-sector or at the midpoint). The front edges of normal offsetting are locked as constraints in this procedure, i.e. these edges are not flipped but refined if they are non-Delaunay.

The algorithm iteratively destroys edges for which one of the following criteria is met:

- the edge is longer than the maximum edge length supplied by the user,
- the opposite angle in a neighbouring triangle is larger than the maximum angle,
- the edge is the longest edge incident at a node that is highly connected, or
- other criteria can be easily added.[1].

These edges are destroyed by refining at the Voronoï centres of the neighbouring triangle with the larger circumradius. After the point in-

[1]In this implementation, the refinement criteria of the generator MESH-ISE [Int99c] are accessible to this algorithm.

sertion the BMCD is reestablished immediately. This reason for the use of the BMCDT at this point is that it guarantees that Voronoï centres lie in the same material region as the triangle they belong to. In some cases multiple points are needed to effectively destroy the edge (Fig. 3.7).

This technique is used because Voronoï centres are places relatively far away from other mesh points. Thus, the result is a graded mesh of good quality [She97b, Rup95]. If the insertion point is too close to an interface or locked edge, this edge is refined first. By observing the constraints the anisotropy of the mesh can be maintained, if the demanded isotropic mesh density is not too small. The method of inserting Voronoï centres cannot, however, create or enhance any anisotropy.

The termination of the refinement algorithm is important. For the edge-length criterion the algorithm stops because, in each step, the edges become shorter.

When suppressing large angles, convergence can be proved for a limit of 120^o [HR00]. In practical examples the maximum angles can be pushed below 100^o. The goal of *obtuse-angle-free* meshes, however, cannot be obtained with unstructured meshes. As the only method the quad-tree can achieve this goal [BE92], but only at the expense of many mesh points (with the exemption of some special – axis aligned – cases).

Concerning the suppression of high connectivity, this refinement can destroy the normal offsetting structure if it tries to lower connectivity to under eight edges per node. For a maximum of ten edges per node the algorithm terminates.

3.5 An intersection-test-less method [2D-RECONNECT]

The weakness of normal offsetting, as it is described so far, lies in its intersection test. In this section we describe a alternative method that exchanges the global search by a local search. The running time of this method then grows linearly with the number of edges in the front.

At first, a constraint DT of the boundary mesh is constructed; there

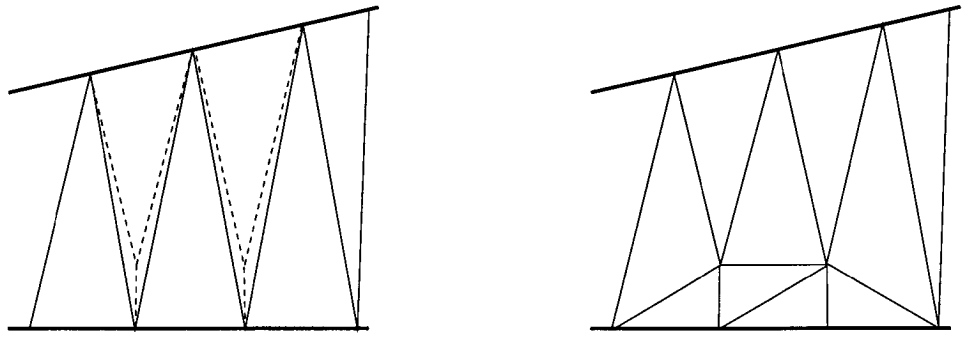


Figure 3.8: *Point insertion with local reconnection*

are multiple methods for doing this, e.g. in Sec. 3.4. Note that this is a much simpler task than triangulating the remaining polygon because this polygon contains more reflex points, which increases the complexity. The front is again initialised by the interface edges and the image points are computed in the same manner as before.

The points of the new layer are inserted into the DT of the boundary mesh, and are locally reconnected in a constrained Delaunay manner. The *local* intersection test now works by checking new points against boundary edges and existing front edges. A point is rejected if it intrudes the diametral circle of a boundary edge (see left of Fig. 3.9). Additionally, if the insertion of this point lets a front edge become non-Delaunay, the point is not inserted. By exploiting the neighbourhood relations these conflicts can be found in a local search.

If the images of the two endpoints of a front ends have been inserted and the connecting edge exists it is locked and cannot be removed anymore by are future reconnection step (see [MW95] and Fig. 3.8). In many cases the new front edge is recovered automatically.

In concave parts of the geometry, where the front is expanding, it may happen that an edge is not recovered automatically, although the end points can be inserted (see right of Fig. 3.9). In that case three methods can be pursued:

- Recover the edge actively by swapping, in which case the Delaunisation will regard this edge as a constraint and insert points. The front continues at the recovered edge.

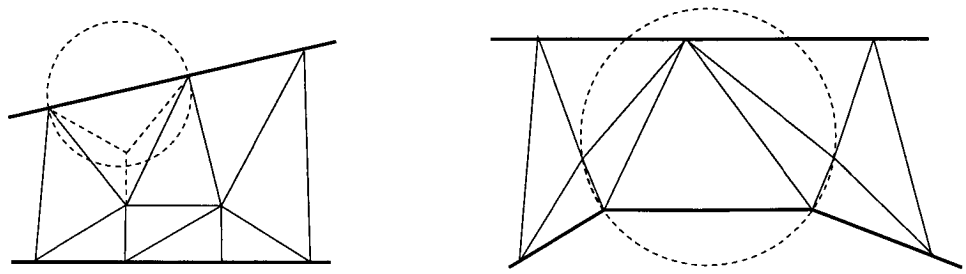


Figure 3.9: *A point that is not be inserted (left), and an edge that is not recovered automatically (right)*

- Recover the edge actively by point insertions, which involves more work to find a good location for these points. In this case the front continues at multiple edges.

- Do nothing. If the edge is blocked by some interface or constraint, the front cannot be continued here.

The implementation of the first approach gives already satisfactory results, in particular as it automatically detects the third case, by simply not being able to recover the edge.

For practical examples the alternative algorithm is not necessarily faster than the intersection test, because the neighbourhood search itself is expensive.

This method has another advantage of being able to produce elements of better quality because it detects proximity easier. If the last layer is very close to an opposing front its edges pass the intersection test, but they form bad quality elements. The alternative method rejects these points.

3.6 Boundary grid for Noffset2d [2D-BOUNDARY]

The discussion has so far considered only the volume mesh and the 1D-boundary grid was taken for granted. But the quality of the volume

mesh naturally depends on the boundary mesh it started from. The main algorithms to discretise the polygonal boundary description are:

- edge length criterion: this algorithms refines a boundary according to a (local) edge length.
- balancing: neighbouring edges should not differ in length by more than a factor of two.

3.6.1 Thin layer regions [2D-THIN]

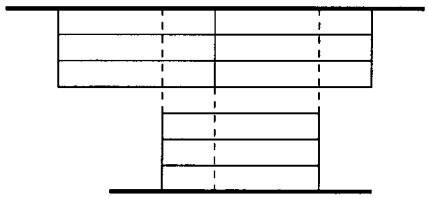


Figure 3.10: *Point propagation at non-matching stacks of anisotropic elements: the dashed lines are introduced by the Delaunisation*

At thin layers a special treatment is necessary. A layer is regarded as *thin* if the fronts starting from opposite sides of the layer are not yet isotropic (due to the coarsening) when the fronts collide. The problem is illustrated in Fig. 3.10: the points in the last layer are so close to the edge of the opposing front that the front edges are not Delaunay. In consequence the Delaunisation inserts a series of points, with the result of a lowered anisotropy and an unbalanced mesh. If the last layers were almost isotropic this propagation could be terminated after a single point.

In order to quantify the problem, consider the following estimation: let d be the layer thickness, h the thickness of the first layer and f the coarsening factor. After n layers the two fronts meet approximately at

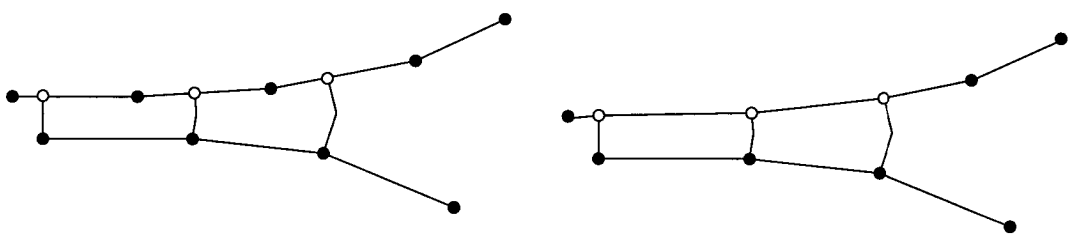


Figure 3.11: *At thin layers boundary points must be arranged that growth lines of normal offsetting meet.*

Algorihm 3.2: *Thin Layer Algorithm*

INPUT: boundary grid, regions A,B,C that sandwich a thin layer in region B, tolerance for geometric changes	
	find nodes at interface $AB \rightarrow listAB$ find nodes at interface $BC \rightarrow listBC$ that constitute the thin layer
	map points of $listAB$ to interface BC (mirror at medial axis)
	remove points of $listBC$ from the mesh if the geometry change is smaller than tolerance
OUTPUT: matching boundary grid	

the middle of the layer:

$$\frac{d}{2} \approx h \sum_{i=0}^{n-1} f^i = h \frac{f^n - 1}{f - 1}.$$

If the final layer is isotropic then for the boundary edge length l the approximate relation holds

$$l > h f^n.$$

From this it follows that for thin layers the approximate inequality is valid:

$$l > \frac{d}{2}(f - 1) + h.$$

As a remedy, two methods are possible. Firstly, one could refine the boundary according to this inequality in such away that the final layer is isotropic and the propagations do not take place. But too many points are needed for this solution

An alternative is to relocate the boundary points such that they match, i.e. the nodes in the final layers lie close to each other like it is shown in Fig. 3.11. The algorithm is sketched as Alg. 3.2. The main idea is that points from one side are mapped to the other side by means of the medial axis and the old points of the target interface are removed, if the geometry does not change to much. With this *matching* boundary mesh, the points of the final normal offsetting layer do not propagate because

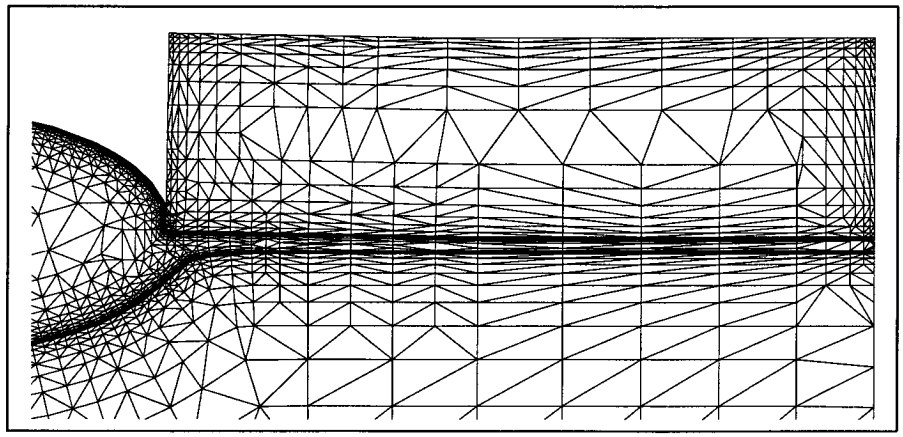


Figure 3.12: *A thin oxide layer in process simulation: the growth lines of normal offsetting seem to go through the oxide. The anisotropy of the elements is conserved and the 1D grid remains balanced*

they do not intrude any circumspheres of triangles of the opposing side. Clearly, such an algorithm changes the geometry if the interfaces are not linear. This has to be done with care, in order not to alter the physics of the problem too much.

A realistic case study is a thin oxide layer in a process simulation: Fig. 3.12 shows the effect of this algorithm.

3.6.2 Limitations of anisotropy in 2D [2D-REFINE-CURVATURE]

The objective of the point placing strategy of normal offsetting is that the anisotropic mesh is Delaunay. At corners this is not necessarily the case if the boundary grid is too coarse. Figure 3.13 depicts a conflicting (left) and the limiting (right) case: if R is far away from P, then P' lies inside the diametral circle of the boundary edge PR. The limiting edge length is

$$\overline{PR} = \frac{2h}{\sin 2\phi}.$$

This local length is taken as an additional criterion to generate the boundary grid, which results in some kind of curvature dependent refinement. From another point of view this result can be seen as a limitation to anisotropy. Since anisotropy is defined as $A = \overline{PR}/h$, the maximum anisotropy at such a corner is

$$A_{max} = \frac{2}{\sin 2\phi}.$$

For straight line segments this formula evaluates to infinity, which means that – locally – there is in this case no restriction to anisotropy due to the BMCD criterion.

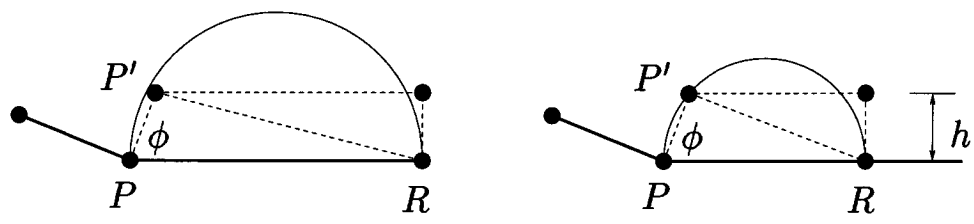


Figure 3.13: *left: the edge PR is so long that the image point P' intrudes its diametral circle; right: the limiting case*


Seite Leer /
Blank leaf

Chapter 4

3D case: Noffset3d

The idea of normal offsetting is carried over to 3D. The node creation and front advancement are discussed. Two approaches are presented for robust extraction of the final mesh. The creation of a volume triangulation is crucial in both methods.

In this chapter the normal offsetting idea is extended to 3D. In the layering prismatic elements replace the quadrilaterals. Some algorithms like local repairs of the front or the intersection test find their equivalents in 3D. Exceptions are, that the parallel character of front faces cannot be preserved, and that the triangulation of the remaining polyhedron cannot be done by a direct method. In fact, the extraction of the final mesh turns out to be a major obstacle towards a robust implementation. Therefore indirect methods are used for this task, two of which are described and compared.

The first method keeps the global intersection test which is typical for a classical Advancing Front mesh generator. An alternative method replaces it by a local search.

4.1 Input to the algorithm

Similar to the two dimensional case the input to this algorithm is a geometric model description and refinement information, but only a triangular surface mesh is needed for this chapter. The construction of such a surface mesh is subject of a later chapter.

The parameters for the 3D generator are defined using the same language as for the 2D version (App. B). Again the main parameters are thickness of the first layer, coarsening factor, maximum edge length, and number of layers. These parameters can be defined locally with some flexibility, e.g. the front need not start from all boundaries.

4.2 A 3D string algorithm [3D-NOFFSET]

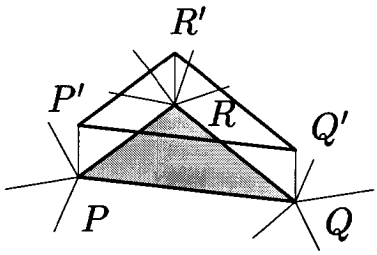


Figure 4.1: *Building block of normal offsetting in 3D: a prismatic element attached to a surface triangle.*

The analogue to 2D quadrilaterals in 3D normal offsetting are prismatic elements (Fig. 4.1) that are added to the triangular surface mesh. In many cases, these elements are not exactly prisms because in general, the triangles they contain cannot be created parallel and of same size; also the lateral faces of prisms are not necessarily rectangles, but quadrilaterals, so they need to be triangulated.

For further understanding, it is important to see that the prismatic elements are only temporary in nature: the mesh extraction algorithm replaces them by tetrahedra. In that triangulation the diagonal introduced into the quadrilaterals can be changed, so that untriangulatable configurations are resolved automatically.

As the front advances local operators are applied to improve quality. Additionally, collisions with boundaries and other parts of the front must be detected: two different methods are discussed.

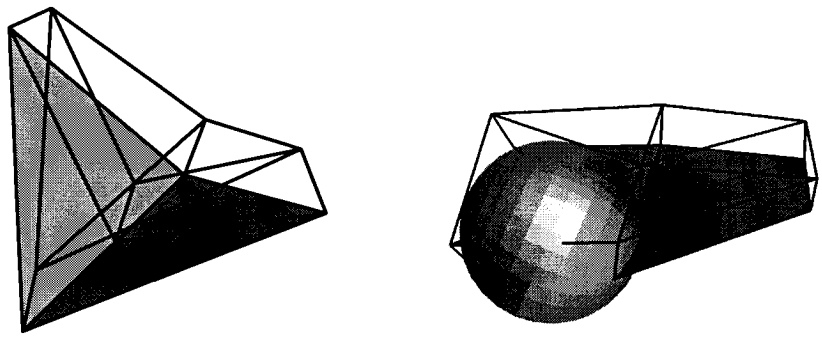


Figure 4.2: *Construction of image layer; left: at folds; right: the image point should not intrude the equatorial sphere of a surface triangle.*

4.2.1 Node creation

The goal of the normal offsetting method is the construction of an image layer of the surface mesh with a given thickness. Additionally, because of the BMCD requirement, the nodes of the image layer should not lie in the equatorial sphere of surface triangles (see right in Fig. 4.2). As a compromise between these contradicting demands the following local construction is chosen.

Let P denote a vertex of the surface mesh and P' its image point. The surface node is attached to surface triangles with k distinct surface unit normals $\mathbf{n}_i$; the marching distance at normal i is h_i. Depending on the number of normals, different computations are chosen.

$k = 1$ In the simplest case $P' = P + \mathbf{n}h$.

$k = 2$ The two planes ($i = 1, 2$) defined by $(\mathbf{x} - P) \cdot \mathbf{n}_i = h_i$ intersect in one line. P' is chosen on this line that is the closest to P, which gives a third equation: $(\mathbf{n}_1 \times \mathbf{n}_2) \cdot (\mathbf{x} - P) = 0$ (see left in Fig. 4.2).

$k = 3$ In this case, there are three equations $(\mathbf{x} - P) \cdot \mathbf{n}_i = h_i$ that need to be solved.

$k > 3$ The system in this case has too many equations to satisfy all conditions. A point is chosen that is as close as possible to all planes by a least-square-fit, i.e. P' minimises the function

$$L(\mathbf{x}) = \sum_i [(\mathbf{x} - P) \cdot \mathbf{n}_i - h_i]^2.$$

The matrix equation to be solved for $\mathbf{x}$ is

$$\sum_i \mathbf{n}_i[(\mathbf{x} - P) \cdot \mathbf{n}_i - h_i] = \mathbf{0}.$$

There are cases where there exists no point that is visible from all faces attached to P. The described procedure does not consider this problem: the only solution would be to create multiple images for this point. Since this problem does not arise often, the implementation does not treat this special case.

The second two cases involve the solution of a three–dimensional problem $M\mathbf{x} = \mathbf{y}$. The matrix M may be singular or almost singular; in that case the solution can either not be obtained or it is not reliable as an answer to the geometrical question due to floating point roundoff. In these cases the following procedure is applied.

The fact that the matrix is singular or almost singular is equivalent to one eigenvalue of M being zero or almost zero compared with the other eigenvalues. Let $\mathbf{v}_0$ be the eigenvector of this eigenvalue, and V the (two dimensional) orthogonal space of $\mathbf{v}_0$ spanned by two unit vectors $\mathbf{v}_1$ and $\mathbf{v}_2$:

$$V := \{\mathbf{v}_0\}^\perp = \mathbb{R}\ \mathbf{v}_1 + \mathbb{R}\ \mathbf{v}_2.$$

Furthermore, Λ defines the projection $\mathbb{R}^3 \to V$ (i.e. $\Lambda^{\alpha\beta} = v_1^\alpha v_1^\beta + v_2^\alpha v_2^\beta$). A reduced problem is solved in two dimensions:

$$\Lambda M_{|V}\mathbf{x} = \Lambda\mathbf{y} \text{ for an } \mathbf{x} \in V.$$

Of course, the 2D problem can still be singular or almost singular. In that case only one eigenvalue of M differs (substantially) from zero. Let $\mathbf{v}_2$ be that eigenvector, then

$$\mathbf{x} = \frac{\mathbf{v}_2 \cdot \mathbf{y}}{\mathbf{v}_2 \cdot M\mathbf{v}_2}\mathbf{v}_2$$

is chosen, i.e. x lies in the subspace defined by $\mathbf{v}_2$.

4.2.2 Data structure

The data structure (Fig. 4.3) consists of the hierarchy of the items node, edge, and face and represents the layer structure. The structure in one

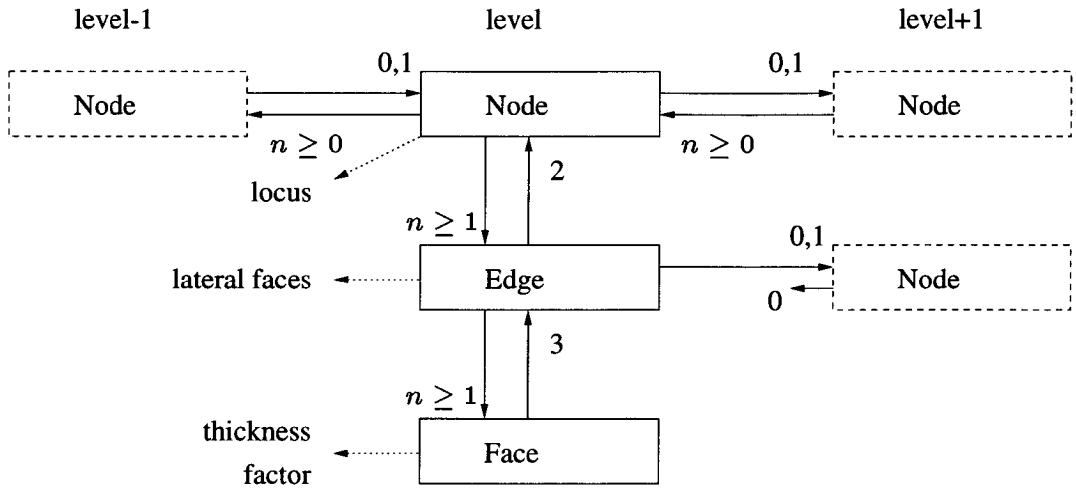


Figure 4.3: *The hierarchical data structure for normal offsetting in 3D.*

layer is such that faces are made of three edges and an edge connects two nodes. Back-links are inserted for fast neighbourhood searches. The connecting links between layers are pointers from nodes to its (unique) child and its (maybe multiple) parents. The nodes in the first layer are characterised by having no parents.

4.2.3 Front advancement

When the front advances, the validity of the generated prismatic elements is checked locally and globally.

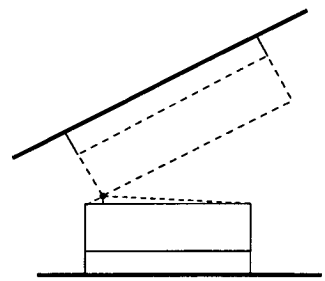


Figure 4.4: *A 2D example for proximity: if the proximity is detected the dashed element should not be created*

Intersection test The global intersection test ensures that the front does not cross itself or the surface of the domain. Like in other advancing-front implementations this is the most expensive module in terms of computational complexity. An octree is employed to speed up the search for possible intersection partners.

Like in 2D, the intersection test is also critical in terms of robustness, as a single intersection that has been overlooked, generates crossing

elements and the entire mesh is invalid. The policy here is to reject a face in case of doubt, because badly shaped elements would be created if elements were allowed that are close but not intersecting (Fig. 4.4). The front is stopped locally if such nearly-intersections are found, but this is not always possible since proximity cannot be measured with such an intersection test.

This global test is inevitable in classical advancing-front approach. Section 4.4 proposes an alternative algorithm that replaces the global search by a local test. It also offers a way to measure proximity.

Quality check Some invalid elements or elements of bad quality can be found by local algorithms. Tests for poor elements detect if

- the edge length in the child level is too short,
- the area of triangle in the child level is too small, and
- the element is twisted. i.e. the direction of the normals differ by more than 90 degree.

These problems can be mended by two operators MERGE and REFINE as shown in Fig. 4.5, that work on the edges of the front. MERGE collapses an edge and creates tetrahedra (instead of prismatic polyhedra) on top of the attached triangles (e.g. for a twisted element in right of Fig. 4.5). The REFINE operator splits an edge and in consequence splits the attached triangles. Polyhedra of higher order are created in this case. The operators are applied on the edge that cause the defect in order to continue the front. In this process, the nodes can be relocated;

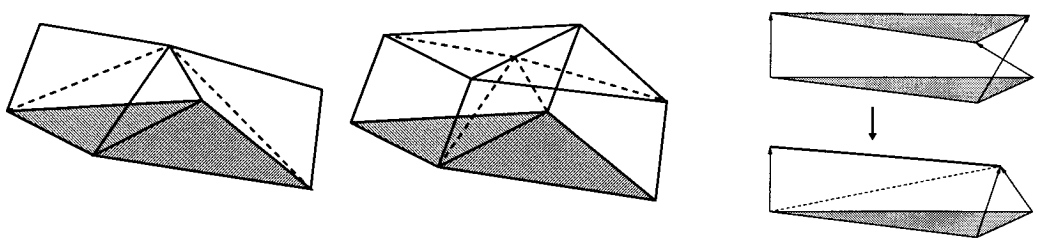


Figure 4.5: *The operators* MERGE *(left) and* REFINE*(middle) work on edges of the front; the dashed lines show new edges.Right: one edge of a twisted element is* MERGE*ed to a point.*

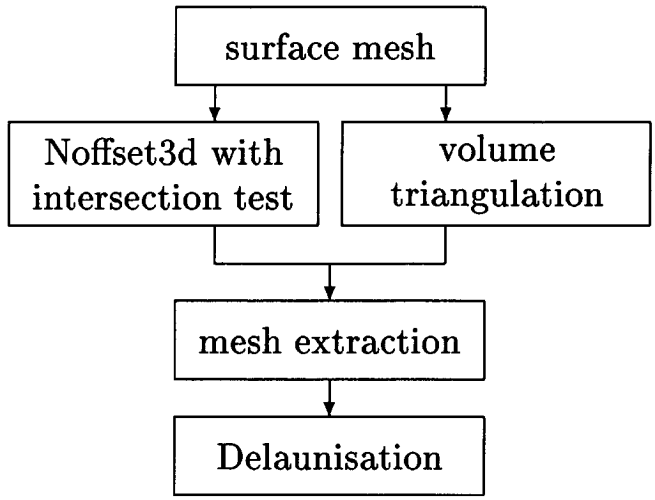


Figure 4.6: *Flowchart of algorithms for a classical advancing-front approach*

so the parallel character is sacrificed to be able to continue the front. If the attempt to repair the new front fails the front is stopped locally.

4.3 Flow I

Two strategies are possible to handle mesh extraction: one method uses a classical advancing-front approach with a global intersection test. The method which is described in Sec. 4.4 uses a Delaunay reconnection algorithm with localised intersection tests.

The flow chart of the first method is depicted in Fig. 4.6. Normal offsetting is applied to the surface mesh as it has been described in Sec. 4.2. For the following a volume triangulation is needed, which may be constructed be any algorithm. Here, a method is presented that does not created any volume points. Using the volume triangulation as a base, the final mesh for the normal offsetting can be extracted and delaunised, as it will be discussed.

4.3.1 Volume triangulation [3D-TRIANGULATION]

The volume triangulation method of this implementation follows the references [She97b] and [KSVF00b]; it uses a randomised incremental Delaunay method. Starting with a cube of three times the size of the bounding box, which is triangulated by five tetrahedra, the vertices of the surface mesh are inserted using an incremental Delaunay technique (Sec. 2.6).

Since the Delaunay kernel does not respect material boundaries not all boundary faces are present in the triangulation and the recovery of interface edges and triangles becomes the crucial part. If an interface triangle is missing the material associations are lost and the geometric model destroyed.

To recover boundary faces point insertions cannot be avoided for which different strategies are known. George et al. [GHS88] place the refinement points *off* the edges and triangles and are therefore able to recover faces in the volume mesh exactly as they are in the surface mesh. However, since those faces violate the Delaunay criterion, they cannot be accepted in the final mesh and are refined in the final Delaunisation step. Therefore, those refinement points are added at an earlier stage, and edges and faces are recovered as their refined entities. The algorithm follows Shewchuk's work [She97b] with the modifications that it does not fail on small inputs.

At thin layers the algorithm terminates – in the worst case –, when an isotropic triangulation is reached; then, the equatorial spheres of triangles on one interface do not contain points of the other interface. This is not the desired goal of this mesh generator, but if the surface meshes of the two interfaces are not compatible, an isotropic triangulation is the only way to terminate given the requirement of a BMCDT.

4.3.2 Constrained incremental Delaunay kernel [3D-INCREMENTAL]

The mesh extraction algorithm needs a helper algorithm, the *constraint Delaunay kernel* (CDK), which is an extension of the standard Delaunay kernel (Sec. 2.6). The constrained kernel leaves the material boundaries

Algorihm 4.3: *A constrained star-shaped Delaunay kernel (CDK)*

INPUT: $\mathcal{T}$ in $\mathbb{R}^3$, a point P, and (maybe multiple) $t_i \in \mathcal{T}$ with $P \in \bar{t}_i$	
	initialise cavity:
	$C(P)$ with t_i and $\partial C(P)$ as the faces bounding $C(P)$
	build maximal cavity:
	`for all` $f \in \partial C(P)$ and not boundary or constraint
	$n =$ neighbouring tetrahedron of f with $n \notin C(P)$
	`if` P inside circumsphere of n
	`then` add n to $C(P)$ and update $\partial C(P)$
	(do not remove constrained faces from $\partial C(P)$)
	`until` all $f \in \partial C(P)$ are boundary, constrained or no circumsphere intruded
	correct cavity:
	`for all` $f \in \partial C(P)$
	`for all` neighbours $n \in C(P)$ of f
	find normal direction v of f in n
	if P is not visible from f wrt. v remove n from $C(P)$ and update $\partial C(P)$
	`end-loop`
OUTPUT: star-shaped cavity $C(P)$ that respects constraints	

and constraints intact that the full Delaunay kernel destroys. A constraint is a locked face, e.g. a front face of normal offsetting, in the sense of a constraint-DT, which was introduced in Sec. 2.4.

Algorithm 4.3 sketches the CDK algorithm. Given the insertion point and the seed tetrahedron that contains the point, it expands the cavity like in the standard Delaunay kernel, but without crossing material boundaries. In order not to loose the constraints those faces are kept in the data structure that describes the outline of the cavity, even if the two neighbouring tetrahedra are both inside the cavity. At this point the cavity with the constraint is not star-shaped anymore.

To reestablish star-shapedness, the correction algorithm by George and Borouchaki [GB98] is used, which was originally intended to overcome numerical problems of the in-sphere-test. It shrinks the cavity by removing tetrahedra from the cavity that are neighbours of faces invisible from the inserted point, although their circumsphere was intruded by this point. For constraint faces, both sides are tested for visibility, so in any case one of the neighbouring tetrahedra is removed for the cavity. The constraints are therefore on the outline of the cavity or outside, when the correction algorithm finishes. In an extreme case the cavity can shrink so far that it contains only the seed tetrahedron.

The CDK-algorithm leaves the triangulation in a state that is not Delaunay, which can be accepted temporarily. Its advantage is that material interfaces can be handled easier.

4.3.3 Extraction of the final mesh [3D-EXTRACTION]

With the two previous algorithms at hand, the volume triangulation and constraint Delaunay kernel, the mesh extraction works a follows. The points created by the normal offsetting are inserted into the volume triangulation with the help of the CDK. For each point in the first layer the seed tetrahedron can be found in the neighbourhood of the surface point of which it is the image. The seeds for points in the subsequent layers can be found by following the growth lines of normal offsetting. The CDK takes care to generate a mesh that is locally optimised, compared with the mesh that would be created by simply splitting the seed tetrahedron. Additionally, the CDK does not remove interfaces like the standard Delaunay kernel.

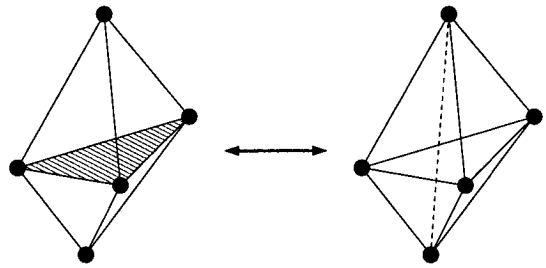


Figure 4.7: *The basic (mutually inverse) transformations for mesh connectivity in 3D:* 23*-flip and* 32*-flip.*

In addition to the points, also the front faces of normal offsetting are enforced in the extracted mesh in order to preserve the layer structure. The recovery of the front faces consists of simply identifying these faces if they are present in the Delaunay triangulation. It is the aim of the mesh generator to create as many of these triangles as possible. Other edges and faces are recovered by swapping faces or edges. The basic transformations that are applied are the *23*– and the *32*–flip (Fig. 4.7). The first destroys a triangle and the second an edge that is shared by exactly three faces. Both transformation can only be applied if the outline is convex, and are inverse to each other. A face that has been identified or recovered is locked and the CDK treats it as a constraint for subsequent point insertions. Not all faces can be recovered this way, but to avoid over-refinement, the requirement to recover *all* parallel faces is not maintained.

Concerning the mesh extraction, the 3D algorithm differs largely from the 2D case. In 2D a direct triangulation is always possible without additional points. The 3D task of extracting a mesh for the void that is not touched by normal offsetting could be formulated as triangulating a remaining polyhedron. This problem cannot be solved without adding points to the polyhedron and algorithms that try to terminate with a small number of additional point tend to be not robust. Ruppert and Seidel [RS92] discuss the problems regarding the triangulation of polyhedra. In the case of 3D normal offsetting it is not necessary to triangulate the remainder in a strict sense. The indirect method that has been developed here is robust and sufficient for this purpose.

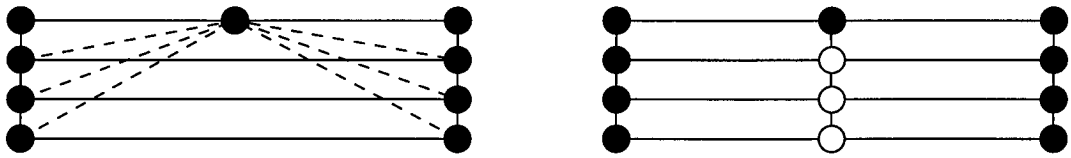


Figure 4.8: *Propagation of a refinement point through anisotropic layers in 2D. Many additional points are inserted (empty circles) and the anisotropy is lowered but it is not completely destroyed.*

4.3.4 Delaunisation [3D-BMCDT]

Since the point insertion (using the CDK) and the recovery of the parallel faces does not follow the Delaunay criterion the extracted mesh has to be converted into a BMCDT. Two different methods can be applied.

Firstly, local transformations (like Fig. 4.7) can be used to successively improve the mesh until it complies with all criteria; this approach has been described extensively in [Vil00]. The front faces generated in normal offsetting are locked in this process, i.e. they cannot be destroyed in a *23*-flip; in cases when this face is not Delaunay it is refined.

Alternatively, the volume triangulation algorithm in Sec. 4.3.1 can be employed with additional input: the points and the front faces created by the normal offsetting method. The front faces are treated as constraints, i.e. like region interfaces, and are recovered by point insertions, if necessary. A weaker condition is applied on constrained faces, since only for interface triangles the equatorial sphere must be point-free in the BMCD condition. In the practical implementation the latter method often terminates with fewer points.

In both cases the front faces are restored in non-Delaunay cases by additional points, which lowers anisotropy to some extent but does not destroy it completely. If one face in an anisotropic stack of faces has to be refined in this way, it is very likely that a neighbouring layer also needs refinement. In consequence a series of points is inserted; this behaviour in anisotropic meshing is called point propagation. The analogue situation in 2D is displayed in Fig. 4.8.

4.4 Flow II

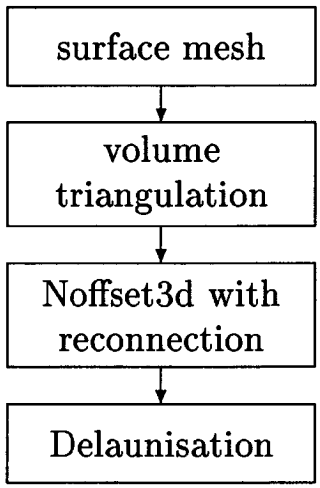


Figure 4.9: *Flowchart for reconnection-based algorithm*

The alternative strategy for the mesh generator extends the idea of an intersection-test-less normal offsetting (Sec. 3.9) into three dimensions. It replaces the global intersection test by a local search and combines the extraction method closely with the point creation [3D-RECONNECT].

The flowchart is slightly different in this case (Fig. 4.9): a volume triangulation for the surface mesh is again needed. But normal offsetting works, in this method, directly with the triangulation. The front is initialised by the interface triangles of each material region separately. The algorithms for node location (Sec. 4.2.1) and local improvement (Sec. 4.2.3) are applied in the same manner as for the classical method. The final mesh is delaunised in the same way.

After each layer is finished, its points are inserted into the volume triangulation with the help of the CDK-algorithm (Sec. 4.3.2). The seed tetrahedron can be quickly found by a neighbourhood search starting from the parent node. The edges and faces that form the new front face are recovered using the *23*– and *32*–transformation in Fig. 4.7. Front faces are not forced by many refinement points in order to avoid over-refinement. Once the face is recovered it is constrained for further actions of the CDK-algorithm and initialised as a front face for the consecutive layer.

Before inserting the image point P' of a node P into the volume triangulation the following conditions are checked. They replace the global intersection test. If any of these conditions is not met, the node is rejected from insertion and the front is stopped locally.

Condition 4.1 P' *is* visible *from* P *if the search path (via neighbourhood relations in the volume triangulation) from* P *to* P' *does not cross interface triangles or locked faces.*

Condition 4.2 *The point P' is* reconnectable *as child of P if it does* not *fulfil one of the following conditions:*

- *P' is inside the equatorial circumsphere of an interface triangle,*
- *for a constrained face that has not P as a vertex, P' is inside the circumsphere of the neighbouring tetrahedron that is not part of the cavity generated by the CDK.*

In other words, P' is *reconnectable* if the insertion into the triangulation, via the CDK, does not let any of the mentioned faces become non-Delaunay. The condition 4.2 fails if the insertion is close to an interface or an opposite front face. Hence, the proximity of P' to an opposite front can be determined by local algorithms.

Whereas the condition 4.1 is checked during the search for the seed tetrahedron that contains P', the fact the P' is *reconnectable* is verified after the cavity has been built.

Another important test is closeness to an already existing point. The points of the seed tetrahedron are candidates: if one is closer than $\alpha = 0.1$ times the local marching distance away from P', then P' is moved to this point.

A final remark on the expected running time of the reconnection algorithm: the time complexity depends largely on the number of elements in the cavity, which can be high in the beginning when no volume points have yet been inserted. Since the points are inserted starting from the surface the size of the cavity does not decrease very fast. Actually, the size of the cavity depends on the geometry: for a cube the cavity contains more element than for an elongated brick.

The final mesh is delaunised by one of the methods described in Sec. 4.3.4.

4.5 Comparison of both methods

The two approaches are referred to as *intersection-test* method and *reconnection* method and are compared in the following fields:

Running time The local search in the reconnection method is linear in the number of faces in the current front, whereas the search in a tree as the time complexity of $\mathcal{O}(n \log n)$. However, the neighbourhood search itself is more expensive than the intersection test. In practical examples the reconnection method proves to be faster.

Memory The intersection-test method needs a tree as an auxiliary data structure. For the reconnection method neighbourhood information must be stored on the faces. Many mesh data structures provide this information already.

Quality The quality of the elements can be bad in the intersection-test method because proximity cannot be detected if the faces do not intersect. The reconnection method finds closeness to the opposite front easily in the neighbourhood search.

Recovery of faces In the reconnection method the front stops if faces cannot be recovered. In consequence some faces are lost in subsequent layers. The intersection-test method starts the recovery in the extraction method, that means the front continues regardless of failure in recovery.

Robustness For both methods the programmer must be careful that no intersection or no point rejection is overlooked. If a Delaunay method is used in the reconnection method, however, conflicts can be found simply be inspecting the the cavity. The consequences of failure are different: the intersection-test method can enter infinite loops, whereas in the reconnection method invalid elements are created, that intersect each other.

4.6 User-defined refinement [3D-REFINE]

Normal offsetting does not fill the void with a fine enough mesh. This can be due to early termination of the front because of intersections, or front faces that cannot be recovered in the mesh extraction, or because the user simply demanded only a limited number of layers. Then, the mesh is further refined isotropically before the final Delaunisation. The criterion is that edges judged too long are bisected; the refinement point is inserted into the mesh with the CDK algorithm described in

Section 4.3.2. In a simple implementation a maximum edges length can be defined on each material region[1].

4.7 Sliver elements

Slivers are tetrahedra of almost zero volume, but having reasonably long edges. They can be part of a DT if four points are almost planar and almost cocircular and if the radius of their circumsphere is limited. Candidates for slivers are created systematically by normal offsetting: at flat parts of the front the quadrilaterals of the prismatic elements can induce slivers, e.g. the four points P, P', Q, Q' in Fig. 4.1.

In an incremental Delaunay method slivers can be suppressed by shrinking the cavity as it was described in the CDK-algorithm. In consequence the mesh is not exactly Delaunay, but the deviation in the Voronoï-diagram is of the order of the thickness of the sliver element. Experience shows that such a small error does not affect the Box Method too much, because the error in the volume of the Voronoï-cells is only small. On the other hand, computation of some properties becomes error prone on sliver elements, e.g. the computation of the circumcentre involves a division by the volume of the element, which can be close to zero.

[1] Other more local criteria are also possible, like defining refinement areas in the manner of MESH-ISE.

Chapter 5

Surface meshing

The generation of surface meshes for normal offsetting is discussed using direct and parametric methods. A mesh optimising algorithm is explained to handle iterative refinement of the surface. A refinement that is adapted to volume normal offsetting is introduced. In addition, a method to construct parametric mapping for triangulated surface patches is discussed.

5.1 Input to surface meshing

For surface meshing, the input description is that of a *boundary representation* (BRep) that is composed of planar polygons as faces. The geometry must be defined as a topological model, so that the connecting edges between two faces are common to both faces. In that respect the input follows the definition of a *piecewise linear complex* (PLC), as introduced by Miller et al. [MTT+96]. In addition to the PLC, the model must form closed volumes for material regions.

Since the surface mesh is the support for the volume meshing, it must be adapted to the volume normal offsetting, which is the goal of

this chapter. In consequence, the same values for the parameters are used for surface meshing.

5.2 Methods

The methods that can be used for surface meshing are limited because the PLC is employed as input description. The reason for using linear input is that often the *real* surface is not known in device simulation or that the geometry itself is topic of the simulation. In other fields, an analytical form, such as Bézier patches, is available to the surface mesh generator and created points can be pushed to the real surface.

In applications such as device and process simulation, where the surfaces are defined by triangulations, some authors suggest to reconstruct a real surface from a given triangulation (e.g. [BF97]). These patches are designed in such a way that they are smooth, close to the original surface, and connect G^1-continuous across edges , i.e. the tangential planes are identical at lines where two patches meet. Walton and Meek [WM96] describe a way to define such patches locally. In consequence, the geometry differs from the polygonal description and the changes can be considerably for thin layer regions; or the layer maybe completely destroyed. For this reason this approach is not followed in this work.

Two different approaches can be used for meshing boundary representations: the direct method refines and derefines the model directly in 3D space. On the other hand a parametric method maps independent patches to 2D and uses 2D algorithms. The difficulty of this approach is the construction of a mapping for patches that defined by triangulations, which is subject of Sec. 5.7.

5.2.1 Direct 3D

One way (see Ref. [BF97] for more detailed description) to generate a surface mesh is to apply transformations iteratively to a given triangulation until the given size and quality criteria are met. The operators used are:

refine edges/faces adds points to the surface,

swap edges optimises connectivity of nodes,

move points optimises element quality, and

suppress edges removes nodes.

These operators, in general, change the geometry of the model. The deviation has to be controlled, so that the final surface mesh differs as little as possible from the input description. In particular controlling deviations only for individual steps is not sufficient because changes can sum up and destroy physical correctness of the description.

This approach can also be used to preprocess a noisy input triangulation to generate a geometric triangulation. To be efficient the algorithms must distinguish geometric features from numerical noise.

5.2.2 Using a parametric map to 2D

For parametric methods the surface is defined by a known bijective function $\mathbf{f} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$. Then a triangulation $\mathcal{T}$ of the domain in 2D can be lifted to 3D via $\mathbf{f}$. In order to make the triangulation $\mathbf{f}(\mathcal{T})$ match certain size and quality criteria in 3D, these criteria have to be translated into 2D-space. This is done by defining a metric in 2D and using an anisotropic method for meshing (see Sec. 2.6). The metric field for 2D can be found by the transformation rules for tensors of differential geometry

$$\mu_{2D}^{\alpha\beta} = \sum_{ij=123} \mu_{3D}^{ij} \frac{\partial f_i}{\partial u_\alpha} \frac{\partial f_j}{\partial u_\beta},$$

or, in the case when an isotropic mesh in 3D with size h is desired:

$$\mu_{2D}^{\alpha\beta} = \frac{1}{h^2} \frac{\partial \mathbf{f}}{\partial u_\alpha} \cdot \frac{\partial \mathbf{f}}{\partial u_\beta}.$$

This metric induces a distance measure in 2D space. Another approach is to interface calls to $\mathbf{f}$, when computing distances between to points

$$d_{2D}(P, Q) = d_{3D}(\mathbf{f}(P), \mathbf{f}(Q)) = \|\mathbf{f}(Q) - \mathbf{f}(P)\|_{3D}.$$

With this method 2D algorithms can be employed. In general the surface of a model has to be decomposed into patches and for each a mapping is found. An implementation has to make sure that the re-meshed pieces fit together when the components are put together again.

5.3 Optimiser for direct surface meshes [25D-TRIANGULATION]

A meshing algorithm using a direct method is explained in this chapter. It uses only operators *refine edges*, *refine faces*, and *swap edges* in planar configurations, so that the geometry model is not altered. For a given triangulation of the surface an optimised triangulation is found by applying these operator for certain edges in a similar way as in the 2D algorithm [Law72].

The conditions that are checked in order to decide whether an edge needs to be flipped or refined are derived from the BMCD criterion:

Condition 5.1 *Let e be an edge with two neighbouring triangles f_1 and f_2; the points in these faces opposite to e are referred to by p_1 and p_2. The edge e is* surface-optimal *iff p_1 is not inside the open equatorial sphere around f_2, and vice versa.*

Condition 5.2 *Let e be an edge with multiple neighbouring triangles f_i; the points in these faces opposite to e are referred to by p_i. e is* surface-optimal *iff for all faces p_i is not inside the open diametral sphere of e.*

Condition 5.2 is tested for non-manifold edges and folds, i.e. where two non-planar faces meet. Condition 5.1 is only applied for edges with *two* planar triangles as neighbours. These conditions reflect the BMCD criterion in 3D: an edge that is not *surface-optimal* cannot be part of a BMCDT. On the other hand, a *surface-optimal* edge may not be present in the 3D BMCDT because the conditions are only checked for neighbouring faces and not globally.

The edges are swapped if they fail to be *surface-optimal* and are *swappable*:

Condition 5.3 *An edge is* swappable *iff it is not locked, it has only two neighbouring triangles, and these triangles are planar (i.e. not a fold edge).*

Some edges may carry a *lock* to protect them from being flipped, e.g. in anisotropic meshing. Folds and locked edges are refined at the perpendicular inter-sector of the disturbing point. In order to avoid spiralling chains of refinement points around corners, these points are protected. A corner in this respect is called an *acute-node* (see also: [Péb98]):

Condition 5.4 *Let P be a node and e_i the fold edges incident to P in sub-domain r. The fold edges are assumed to be cyclically ordered around P. The point P is said to be an* r-acute-node *if for all i holds $\angle(e_i, e_{i+1}) < \frac{\pi}{2}$. A node is an* acute-node *iff there is one region r for which it is an* r-acute-node.

In an initialisation step, these points are identified (once for the rest of the algorithm, because new corners cannot appear), and a local length is evaluated to be a third of the shortest fold edge incident to this corner. Any possible refinement point on such an edge cannot be closer than this distance and – in that instance – it is pushed away.

The conditions are checked for all edges in the surface mesh and action is taken accordingly, until all edges are *surface-optimal.*

5.4 Refining surface meshes

The algorithm in the previous chapter can be localised: it operates only on a limited list of edges. Other edges must be considered as the optimiser changes the mesh but not all edges in the mesh have to be checked in all cases.

Therefore Alg. 5.4 can be used to locally optimise the mesh after refinement. The list of edges is initialised according to the type of problem:

- All edges of the mesh (this is the case a global optimisation).

Algorihm 5.4: *Local optimiser for surface meshes*

INPUT: surface triangulation, stack S of non-surface-optimal edges
while S not empty
pop top of $S \rightarrow e$
if e is surface-optimal **continue**
if e is swappable
then swap e and push edges on S **else** refine e and push edges on S
end-loop
OUTPUT: all edges are surface-optimal

- After refinement of a triangle: the edges on the outline of the original triangle are pushed on the stack (Fig. 5.1 left).

- After refinement of an edge: the edges belonging to the outline of the two triangles and the partitions of the original edge are examined (Fig. 5.1 right).

This optimiser can be used not only for surfaces in 3D, but also in plain 2D and in parametric space that models a 3D surface. The implementation can be unified in the language C++ by using templates. Appendix C discusses the details.

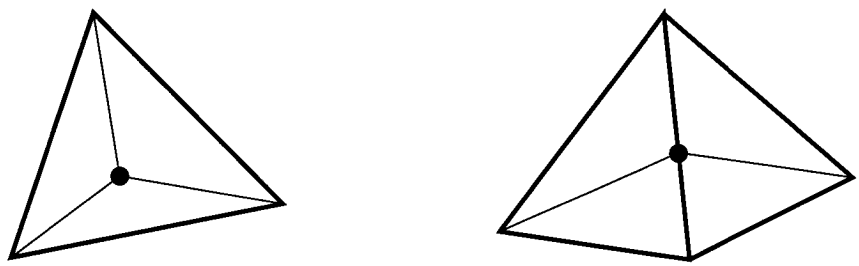


Figure 5.1: *Operators to refine a face or an edge; the solid edges are pushed on the stack and are reexamined.*

5.5 Surface meshes for Noffset3d

With the use of a direct method some refinement conditions can be easily implemented by using the refinement-optimisation algorithm previously introduced. This implementation uses the following criteria to decide which edges to refine:

- A user-defined edge length parameter defined per region or interface [1].
- If an opposing angle is larger than a given value.

This follows the principles of isotropic refinement in 2D (Sec. 3.4). The Voronoï-centres of one neighbouring triangle is inserted if the edge is not a fold or locked. In that case the edge is refined at the edge mid.

5.6 Limitations of anisotropy in 3D

The goal of normal offsetting is to consolidate anisotropy with the BMCD criterion. There are limitations to this, if the surface mesh is too coarse at non-planar patches and the first layer image of a surface node intrudes the equatorial sphere of an attached triangle. A criterion for this is introduced in this section and a refinement strategy that generates surface meshes that conform with this criterion.

The 2D problem was discussed in Sec. 3.6.2; there, further refinement of the boundary was necessary. In a similar way, resolving the conflicts in 3D leads to a special surface meshing technique.

5.6.1 Surface mesh criterion

The following symbols are used in the discussion (see Fig. 5.2)

- P: the surface node,

[1] The refinement criteria defined for the generator MESH-ISE can be used in this implementation.

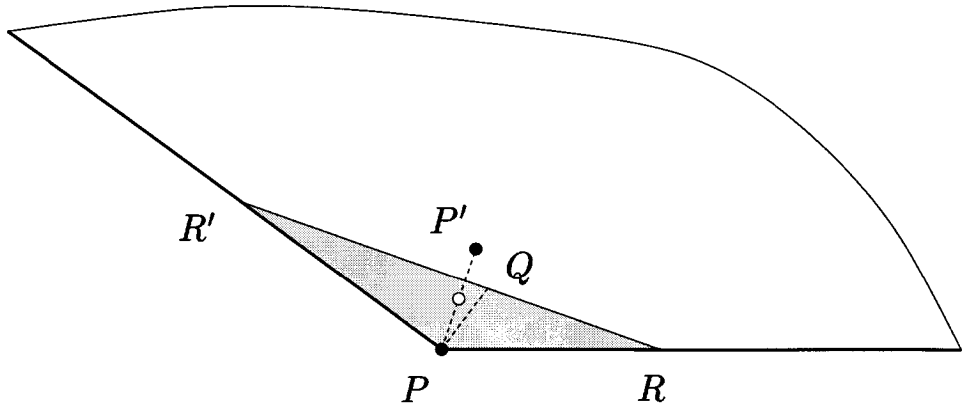


Figure 5.2: *In the shaded area lie the allowed places for centres of equatorial spheres around surface triangles attached to P on this surface patch. The solid lines PR and PR' are folds that bound this patch. The line RR' is the intersection line of the surface patch and the mid-perpendicular plane of the segment PP'. Note that the volume image P' is assumed outside the plane of the paper. Q is the perpendicular inter-sector of P on the line RR'.*

- P': the image of P computed by the method in Sec. 4.2.1,
- f_i: a surface triangle attached to P,
- $\mathbf{n}_i$: the normal vector of f_i, and
- M_i, r_i: the centre and the radius of the equatorial sphere of f_i.

P must fulfil the equation of the sphere $(\mathbf{x} - M_i)^2 = r_i^2$. To comply with BMCD criterion, P' lies outside the sphere:

$$\begin{aligned} (P' - M_i)^2 &> r_i^2 \\ \Rightarrow (P' - P)^2 + 2(P' - P) \cdot (P - M_i) &> 0. \end{aligned}$$

The last inequality can be interpreted as the condition that M_i lies in the half-space defined by the mid-perpendicular plane of the segment PP' on the same side as P. Naturally, M_i lies in the plane of f_i, so another condition has to be fulfilled: $(M_i - P) \cdot \mathbf{n}_i = 0$.

The condition for a *well-refined* triangle attached at this corner can be expressed in terms of its circumcentre M: it must be in the shaded region in Fig. 5.2. In other words:

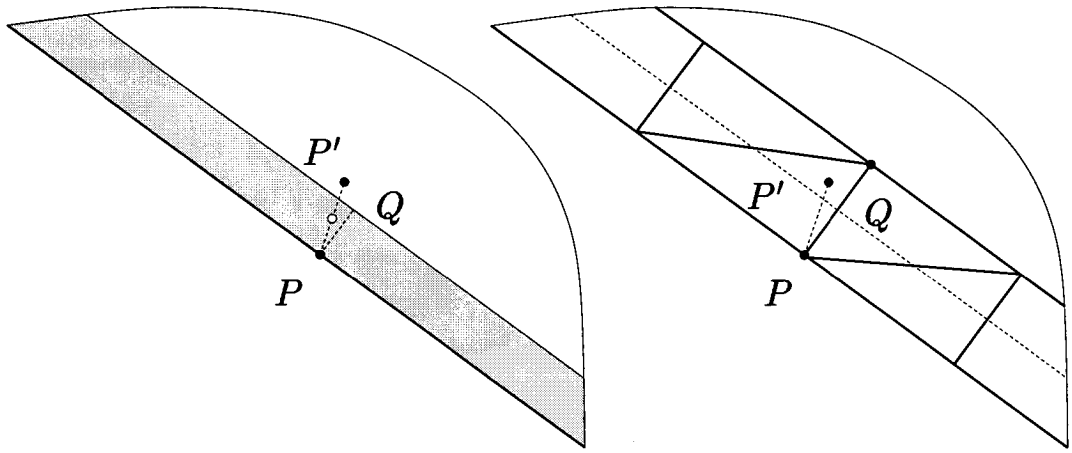


Figure 5.3: *At folds the allowed area for circumcentres is a stripe parallel to the fold. The refinement to generate triangles that are* well-refined *is similar to 2D normal offsetting.*

Condition 5.5 *A triangle f_i attached to a surface point P with image P' is* well-refined *(with respect to a 3D normal offsetting) iff the circumcentre of f_i lies on the same side as P of the mid-perpendicular plane of the segment PP'. The point P itself is* well-refined *iff all its attached triangles are.*

For points on planar patches this is always fulfilled, but not for points at folds or corners.

5.6.2 A suitable refinement algorithm [25D-NOFFSET]

To understand the refinement strategy, the case of a point P on a geometry fold (Fig. 5.3) is examined closer. In that case the mid-perpendicular plane is parallel to the fold, and the allowed region for circumcentres of triangles is a stripe parallel to the fold. The proposed refinement then introduces a mesh line parallel to the fold and at a distance twice the thickness of the stripe. The refinement point is placed at the perpendicular inter-sector of P on this line. In that manner, a *fold-sensitive* mesh is created that looks like a 2D normal offsetting on the surface, which is driven by the 3D volume normal offsetting. There can be other refinement strategies that generate *well-refined* triangles, but the discussed solution fits seamlessly into the concept of normal offsetting.

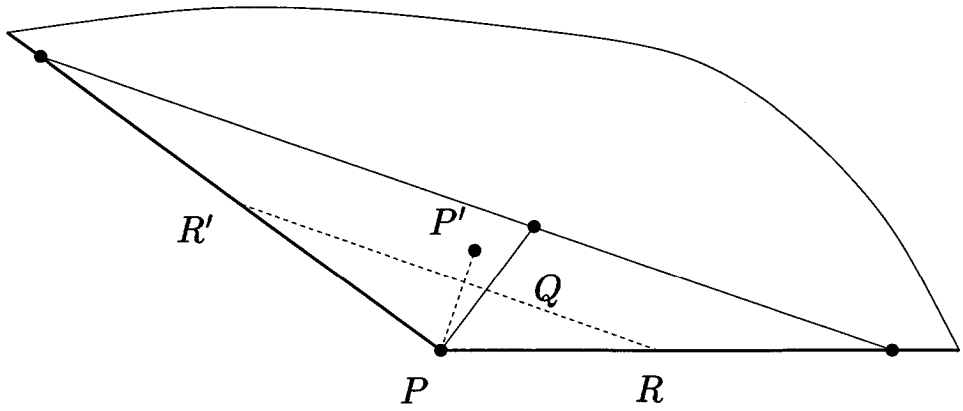


Figure 5.4: *Refinement at a corner, so that attached triangles are* well-refined.

In more general cases, like the one shown in Fig. 5.4, where the mid-perperdicular plane passes through the line RR', the refinement point is chosen on the line that is parallel to RR' that is twice as far away from P. Again the perpendicular inter-sector is taken. In addition the points twice as far away from P as R and R' are inserted on the fold itself. This is necessary, because the image of P in the volume must not intrude diametral sphere of fold edges in the BMCD criterion.

The computation of the refinement points in terms of P, P', and $\mathbf{n}$ is as follows. The point Q on the intersection line and closest to P lies at

$$\begin{aligned} Q - P &= \lambda \mathbf{n} \times [\mathbf{n} \times (P' - P)] \\ \lambda &= \frac{(P' - P)^2}{[\mathbf{n} \times (P' - P)]^2}. \end{aligned}$$

The intersection line cuts the fold edge PS at the point R, which can be found at

$$\begin{aligned} R - P &= \lambda'(S - P) \\ \lambda' &= \frac{(P' - P)^2}{(P' - P) \cdot (S - P)}. \end{aligned}$$

The refinement points are placed, as mentioned before, at twice[2] the distance. The refinement points are inserted into the existing triangulation using the algorithm in section 5.3. Before inserting, closeness to

[2]To avoid problems with roundoff errors, the implementation uses a factor $\alpha < 2$.

existing points or folds is checked and, in that instance, a refinement point closer to P is chosen or the refinement point is rejected.

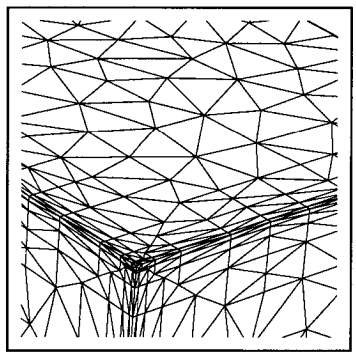


Figure 5.5: *Surface mesh for Noffset3d at the corner of a cube.*

So far, only one layer of a surface normal offsetting is created. The following layers are found by further continuing the growth lines and increasing the step size by the coarsening factor.

The implementation of the advancement follows the principles of the 2D intersection-test-less method, that was described in Section 3.5 for 2D [25D-RECONNECT]. In practice only a few layers are needed.

Figure 5.5 shows the effect of surface normal offsetting for the corner of a cube: the surface mesh lines are parallel to the folds and the transition to an isotropic mesh inside the patches is smooth. In the vicinity of the corner itself the mesh is finer than at the folds away from the corner.

5.7 Construction of the parametric map [25D-PARAMETRIC-MAP]

This section describes a method to solve the problem of constructing a parametric map for a surface patch that is defined by a triangulation $\mathcal{T}$. The limitation is that the surface patch has a boundary that is one closed line, i.e. patches with holes and close surfaces are excluded. To treat surfaces like spheres or tori, these patches have to be divided into parts.

To summarise the method, it takes a mapping of the boundary of the patch to a convex polygon in $\mathbb{R}^2$ as input and solves a linear system to compute an extension of this mapping for interior nodes. In that manner an image of the 3D triangulation is constructed in 2D space, which is a triangulation with the same topology. The parametric map is then defined by linear interpolation on the 2D triangulation. For non-convex polygons the resulting triangulation may have inverted elements, which explains the restriction to convex polygons.

The described method is similar to [Suz90], but offers more general ways to define the matrix.

Input The points in the defining triangulation of the patch are classified into three groups: interior nodes $Q = \{q_i\}$, corners $C = \{c_i\}$, and other boundary points $B' = \{b_i\}$. The distinction between C and B' is defined by the caller of this algorithm. For instance, feature points should be chosen, i.e. where the interior angle is small; then distortion of the 2D map can be avoided best. The set of all boundary nodes is called $B = B' \cup C$ and the union of all points in $\mathcal{T}$ is $M = B \cup Q$. On C is defined $\mathbf{g}_{|C} : C \to \mathbb{R}^2$, which represents the convex polygon that is mentioned in the introduction.

Goal The goal can be now formulated to find an extension $\mathbf{g} : M \to \mathbb{R}^2$, that results in a parametric map with minimal distortions.

Boundary On B' the values of $\mathbf{g}$ can be computed using the arc length measure. Let $c_i = b_{i_\alpha}, \ldots, b_{i_\beta} = c_{i+1}$ be the ordered list of boundary points between two corners, starting with $\tau(b_{i_\alpha}) = 0$ the other $\tau(b_i)$ are calculated by

$$\tau(b_{i+1}) = \tau(b_i) + \|b_i - b_{i+1}\|.$$

Finally, the function $\mathbf{g}$ takes interpolated values on these boundary nodes:

$$\mathbf{g}(b_i) = \frac{\mathbf{g}(c_i)(\tau(c_{i+1}) - \tau(b_i)) + \mathbf{g}(c_{i+1})\tau(b_i)}{\tau(c_{i+1})}$$

Inner points A matrix $A : \mathbb{R}^{|M|} \to \mathbb{R}^{|Q|}$ is needed that will be explained in more detail later. It is assumed that a linear system

$$0 = A \begin{pmatrix} g^\alpha(m_1) \\ \vdots \\ g^\alpha(m_{|M|}) \end{pmatrix}$$

is solved for $\alpha = 1, 2$. The domain is split into B (where $\mathbf{g}$ is known) and Q (where $\mathbf{g}$ is to be determined). The equation decomposes in the

following manner:

$$0 = A_{|B} \begin{pmatrix} g^\alpha(b_1) \\ \vdots \\ g^\alpha(b_{|B|}) \end{pmatrix} + A_{|Q} \begin{pmatrix} g^\alpha(q_1) \\ \vdots \\ g^\alpha(q_{|Q|}) \end{pmatrix}$$

$$\begin{pmatrix} g^\alpha(q_1) \\ \vdots \\ g^\alpha(q_{|Q|}) \end{pmatrix} = -(A_{|Q})^{-1} A_{|B} \begin{pmatrix} g^\alpha(b_1) \\ \vdots \\ g^\alpha(b_{|B|}) \end{pmatrix}$$

with a quadratic matrix $A_{|Q}$. The restricted matrices $A_{|Q}$ and $A_{|B}$ operate on the respective subspaces. In this implementation an simple direct linear solver is used to solve for two 'right hand sides' simultaneously.

Matrix In fact several methods can be used to construct A. For the different approaches, the element matrices A^t_{ij} are given for a triangle t with vertices $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3$ and area $|t|$. The global matrix is then assembled in a loop over all triangles t : $A = \sum_{t \in \mathcal{T}} A^t$.

- One possible matrix is inspired by the Finite Element Method (FEM) solution of Laplace's equation ($\triangle \phi = 0$) in 2D. The coefficients are

$$A^t_{01} = A^t_{10} = \frac{(\mathbf{x}_2 - \mathbf{x}_1) \cdot (\mathbf{x}_0 - \mathbf{x}_2)}{|t|}$$

$$A^t_{00} = \frac{(\mathbf{x}_2 - \mathbf{x}_1)^2}{|t|},$$

  plus cyclic permutations.

- A second idea simulates springs between connected nodes where the strength of the springs is determined by the distance in 3D space. This leads to a Lagrangian function that is minimised:

$$L = \sum_{i,j,\exists \text{edge}(i,j)} \frac{(\mathbf{u}_i - \mathbf{u}_j)^2}{\|\mathbf{x}_i - \mathbf{x}_j\|^2}.$$

  The coefficients are

$$A^t_{01} = A^t_{10} = -\frac{1}{\|\mathbf{x}_0 - \mathbf{x}_1\|^2}$$

$$A^t_{00} = \frac{1}{\|\mathbf{x}_0 - \mathbf{x}_1\|^2} + \frac{1}{\|\mathbf{x}_0 - \mathbf{x}_2\|^2},$$

plus cyclic permutations.

- Another possibility, which is interesting for its simplicity, uses no geometry information, only the connectivity of the nodes:

$$A^t_{01} = A^t_{10} = -1$$
$$A^t_{00} = 2$$

plus cyclic permutations. This approach is similar to the second if the springs are assumed to have equal strength.

The danger of this method is that it may generate inverted elements. The reason to exclude non-convex polygons is that inverted elements can be created in the vicinity of reflex corners. A new reference [SdS00] removes the problem by computing the angles of the 2D-triangulation instead of the node positions. The cost is that a non-linear problem has to be solved. Also, as the 2D boundary polygon is a by-product it is not guaranteed that this polygon does not overlap globally.

Re-meshing of surface patches

The parametric mapping is applied in the module [3D-ISOSURFACE], which incorporates isosurfaces of certain data functions (like the p-n–junction) into the boundary model. In this way the these surfaces are available to the mesh generator like material interfaces. This module takes as input the data defined on vertices of a triangulation. This can be the mesh of a previous simulation or time step, but it can also be generated for this purpose; in such a case the data has to be resolved fine enough but the mesh needs not to be fit for a simulation. The isosuface is then found by interpolation on this auxiliary triangulation. Since this mesh still carries the footprints of the auxiliary triangulation, and since it does not necessarily meet the required mesh density, it has to be re-meshed before incorporating it into the model description. The parametric method is employed in this implementation. For each connectivity component of the isosurface, a mapping $\mathbf{g}$ is computed using the method in Sec. 5.7. An unstructured method generates a mesh for the planar polygon in 2D. To compute the correct distances between

points the function $\mathbf{g}^{-1}$ is used. This function is found be linear interpolation on the 2D mapping of the surface patch. A quadtree structure for searching is used to find the triangle in which a query point lies.

In order to capture all geometric features is is important to measure distances between a re-meshed structure and the patch definition. As an estimate edge mid points and centroids of triangles are computed. Let x_i be the points of the edge or the triangle, and $<.>$ the averaging operator, then an estimate of the deviation for an entity is calculated by:

$$\delta = \|\mathbf{g}^{-1}(<x_i>) - <\mathbf{g}^{-1}(x_i)>\|.$$

The points for the re-meshed surface are created and inserted into the triangulation of the polygon with the reconnection-optimisation approach of 5.3 but in planar 2D with a 3D in-sphere-test (via $\mathbf{g}^{-1}$). Points are created at edge-mids or centroids of triangles; the following criteria are applied:

- Edges that are too long or deviate too much according to the δ-measure are refined.
- Triangles that deviate too much according to the δ-measure are refined.
- Edges that are too short are suppressed.

In the final step the re-meshed surface patch is inserted in the boundary model. Here, care has to be taken that connected faces are cut at exactly the same points so that a the compound model is conforming.


Seite Leer /
Blank leaf

Chapter 6

Examples

The implementation of the software developed in this work was done in the language C++ and compiled for various Unix platforms. The examples were run for timing measurements on a SUN Sparc-Ultra-30 workstation with one 250 MHz CPU and 1152 MB of memory.

2D: Trench IGBT

The trench Insulated Gate Bipolar Transistor (IGBT) considered here is a power device with a curved Si/SiO_2 interface and a long substrate body (See Fig. 6.1). The height of the device is 100 μm and the simulation domain is 5 μm wide.

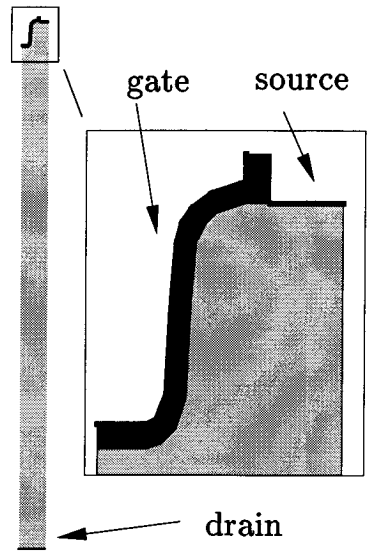


Figure 6.1: *Model for an Insulated Gate Bipolar Transistor (IGBT).*

The current flow in this device is such that the current is confined in a small channel beneath the Si/SiO_2 interface. In order to resolve this boundary layer behaviour, a mesh resolution < 0.01 μm orthogonal to the current flow is necessary. In longitudinal direction a coarser mesh can be accepted. The normal offsetting is able to generate

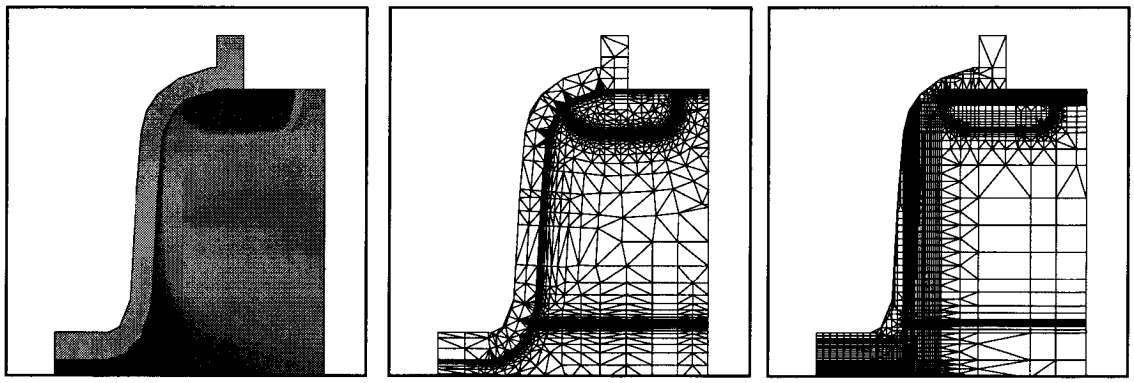


Figure 6.2: *Current density (left) and meshes for an IGBT structure. Normal offsetting mesh(centre) and the coarse quadtree mesh (right) have roughly the same number of nodes.*

such a mesh.

The displayed mesh has in total 3587 nodes in 6104 elements (triangles and rectangles) and was generated in 9.4 s. This time includes computing a temporary mesh, on which the data functions are evaluated. The p-n–junction of this device is computed on this mesh, and incorporated into the boundary.

This example shows not only the layering at interfaces, but also at the p-n–junctions. The coarsening factor has the effect of allowing a graded transition from the boundary layers to an isotropic and coarse volume mesh. The mesh in the gate oxide is a simple unstructured mesh.

The device simulation was performed using the simulator DESSIS-ISE. The resulting total current density is displayed to the left of Fig 6.2: the dark areas have a high current density, the current confinement is obvious. In parallel, the device was simulated with two quadtree-based meshes (generated by MESH-ISE); one of these having roughly the same

mesh type	# of points	# of elements
normal offsetting	3587	6105
quadtree-coarse	4341	4959
quadtree-fine	22507	24192

Table 6.1: *Mesh sizes for the three compared meshes.*

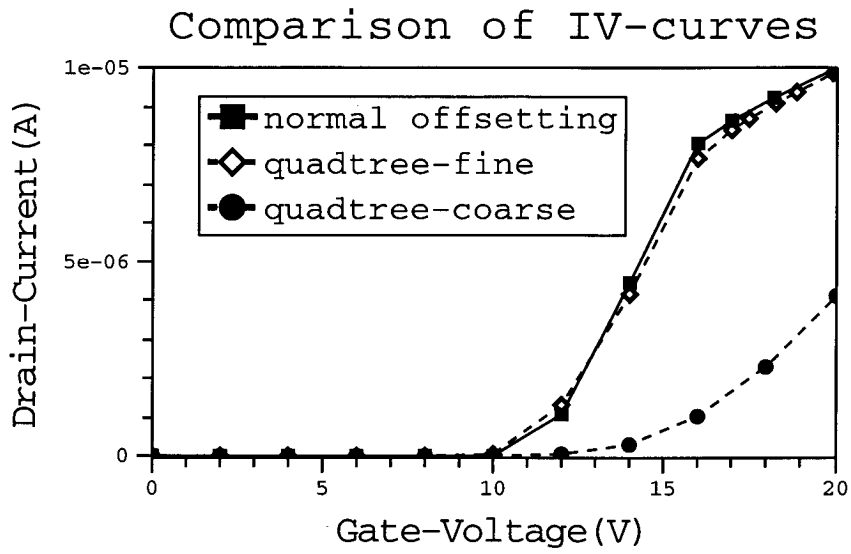


Figure 6.3: *Comparison of IV-characteristic for three different meshes. A much finer mesh is needed in the quadtree-method than with normal offsetting*

number of nodes as the normal offsetting mesh. For this mesh the simulated drain current turns out to be too small. The second quadtree-based mesh is chosen such that the channel is refined enough to resolve the current refinement in the channel correctly. The limitations of quadtree methods are that anisotropic elements are always axis-aligned and that the users can specify refinement criteria only in axis-aligned rectangles. In consequence more mesh points are needed for a correct simulation. In this example six times more points are needed with the quadtree-method than with normal offsetting.

The IV-characteristic is shown for all three meshes in Fig. 6.3. Table 6.1 details the number of points and elements for these meshes.

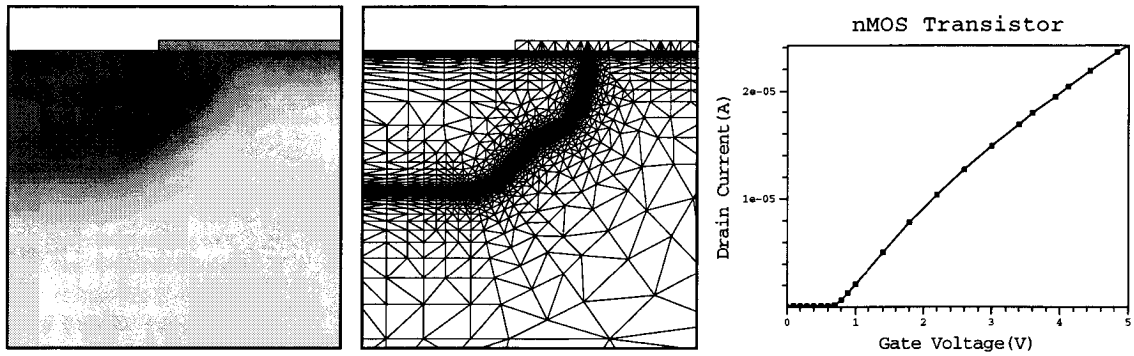


Figure 6.5: *Current density (left) and mesh (centre) for an NMOS transistor structure with junction refinement. The simulated IV-curve is shown to the right.*

2D: nMOS with junction refinement

This example shows how this generator behaves for axis-aligned structures, like this simple model of an nMOS transistor (Fig. 6.2). The simulated area is a 3 μm $\times$ 3 μm silicon square. The gate oxide is 0.025 μm thick. Including the computation of the p-n–junction the mesh generation takes 16.9 s for 6239 nodes in 12251 elements (triangles and rectangles). Again, the simulated current density is displayed in the figure (left) and the terminal drain current is plotted as a function of the applied gate voltage (right).

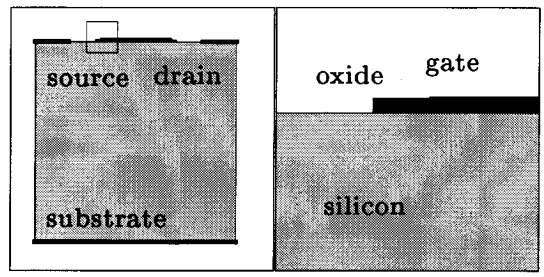


Figure 6.4: *nMOS model*

In the region between the p-n–junction and the source contact the collision a the two fronts is resolved properly and the transition is handled smoothly.

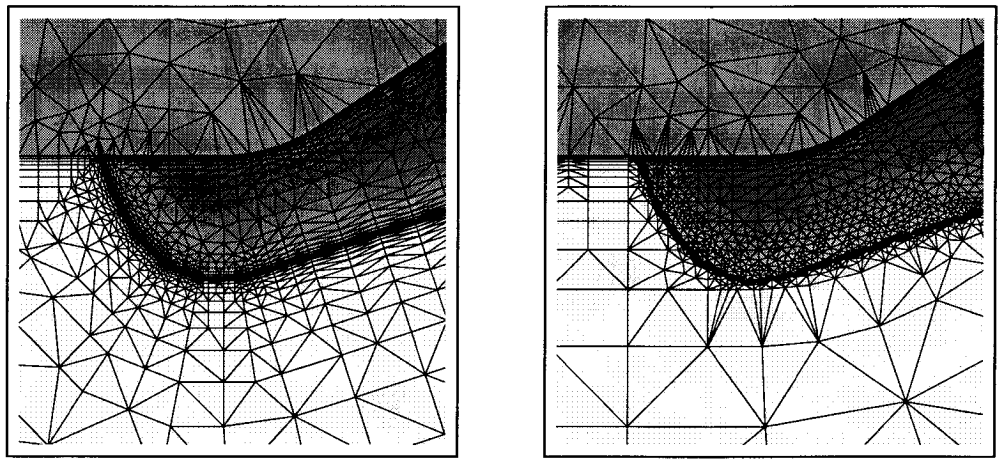


Figure 6.6: *left: normal offsetting started from an isoline that was incorporated into the geometry definition; right: Refining the p-n–junction as a post-processing step, the layers are valid outside the refinement region.*

2D: User refinement

The figures in this section show, for the same model, the various algorithms that control the refinement. Firstly, it is necessary for some simulations to resolve the p-n–junction with a fine mesh. The junction is, by definition, the line where the donor and acceptor concentrations balance each other; in this area the recombination rate of carriers in a forward biased-device is high.

In the framework of normal offsetting two approaches are possible to resolve the junction. On the one hand (Fig. 6.6 left) the junction is computed as a geometric line and added into the boundary representation. Thus, a front is started from the junction like from a material interface. Since the data is in general not defined analytically in such a way that the junction can be found easily, the data is evaluated on an auxiliary triangulation. The junction is then calculated by interpolation. This procedure can be generalised to find and incorporate isoline of any data function (module 2D-ISOLINE).

A refinement approach takes advantage of the fact that the doping gradient at the junction is high. It can be applied to any triangulation because it simply refines edges that exhibit a large gradient. It uses the module 2D-REFINE for refinement, i.e. Voronoï centres are chosen as refinement points. Therefore this refinement is always isotropic. It does

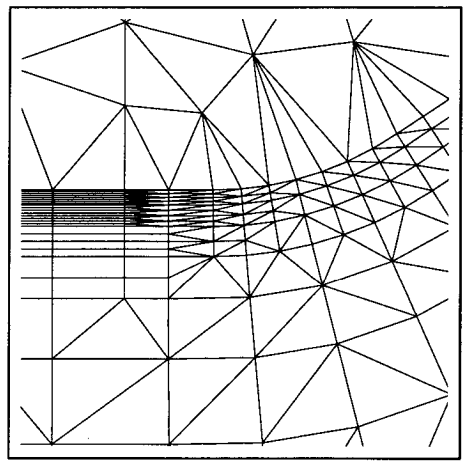


Figure 6.7: *Transition between fine and coarse parallel layers by subdividing elements.*

not, however, destroy any existing anisotropy if this refinement does not require a much finer mesh (Fig. 6.6 right).

The latter method is more general than isoline computation, and it is faster, because no auxiliary triangulation is needed. But, for the stated reason, it cannot produce anisotropic meshes and the transition to the coarse volume mesh can be abrupt.

The main parameters for normal offsetting are the thickness of the first layer and the coarsening factor, which can be defined locally. If the areas in which fine mesh and coarse mesh meet are close to each other, the transition elements have a bad quality. Using the subdivision algorithm (Sec. 3.2.2) it is possible to have areas of different fineness close to each other, if their marching distances differs by powers of 2. Figure 6.7 shows this algorithm in action. In the left part the layers are subdivided twice (giving four layers). A transition area is observed in which the layer is only subdivided once.

2D: Oxidation

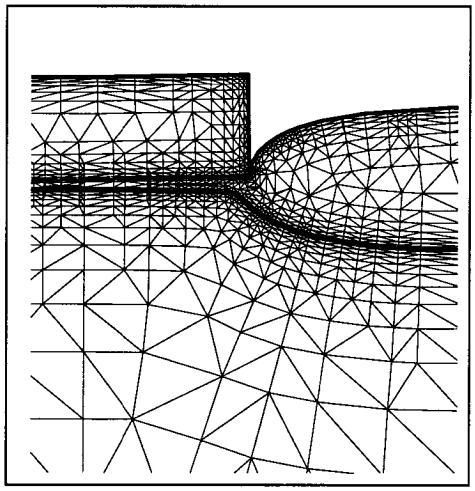


Figure 6.8: *LOCOS simulated with normal offsetting meshes at interfaces.*

For the use in a process simulator the mesh generator must robust, automatic, and fast because many re-meshing steps are needed. The 2D version of normal offsetting has been integrated into the process simulator DIOS-ISE. The fabrication of a LOCOS (LOCal Oxidation of Silicon) structure was taken as a benchmark. The growth was simulated with a state-of-the-art visco-elastic model [PZSF00]. Since no moving grid is available a re-meshing is necessary after each time step. This is a test for robustness of this generator. Figure 6.8 shows the final result and the last mesh of the simulation

The entire simulation takes 9 min, of which 5 min are spent on the generation of 36 meshes. For each mesh, the generation takes between 6.0 s and 14.0 s, on average 8.8 s. The mesh sizes varies between 3689 and 6582 points.

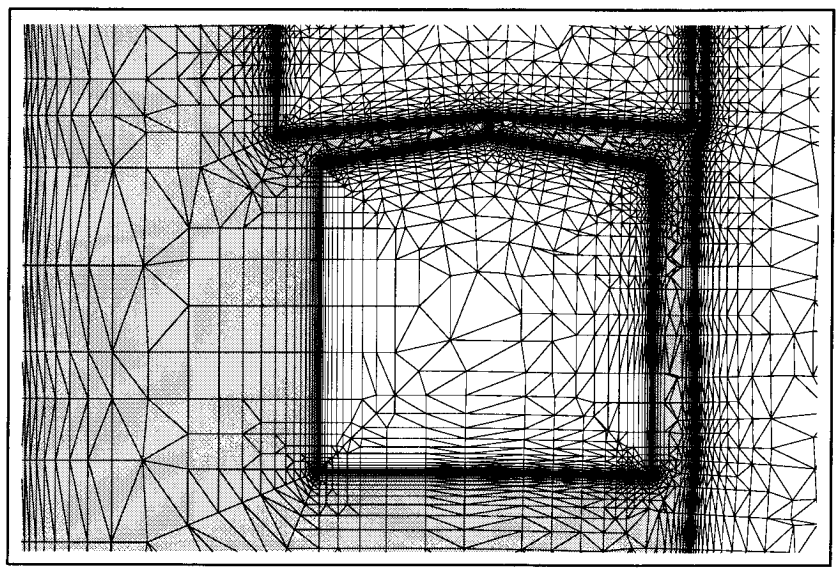


Figure 6.9: *complicated process simulation*

For moving parts of the boundary the process simulator computes the displacement for the time step for each interface node . The simulator needs an interface-adapted mesh to calculate a smooth variation of the displacements. Especially, the mesh size should be balanced along the interface. The normal offsetting meshes perform well in this respect.

Figure 6.9 shows the mesh from an entire process flow that was simulated with normal offsetting meshes. It includes several implantations, diffusion, deposition, and oxidation steps.

3D: An ECL bipolar device

This 3D example is an ECL (Emitter Coupled Logic) transistor. It contains of a silicon brick (8.2 μm $\times$ 4.0 μm $\times$ 3.5 μm) with an L-shaped trench, which is filled with oxide (Fig. 6.10). The contacts are on the top of the device; the collector is separated from the base and emitter contacts by the trench. The bottom face carries the substrate contact. The normal offsetting parameters are such that the front is started from the top and from the bottom, but not from artificial vertical boundaries and not inside the oxide region. The initial thickness is set to 0.02 μm. The final mesh contains 16255 nodes in 91051 elements, and the mesh generation takes 700 s.

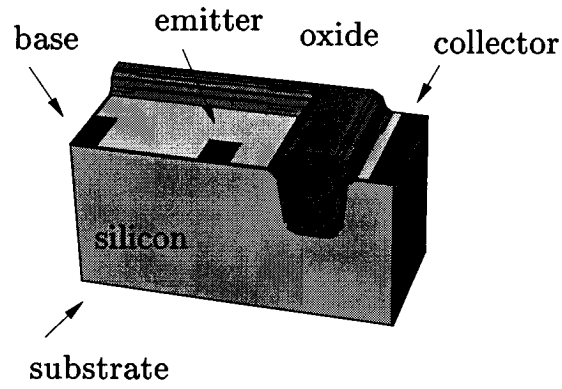


Figure 6.10: *Emitter Coupled Logic transistor*

The right of Fig. 6.11 shows the interior of the mesh and how far the extend into the volume. The remaining volume nodes were generated by the isotropic refinement module (3D-REFINE). The mesh lines follow nicely the interfaces; it also shows problems of this method to recover faces in non-convex corners.

The device characteristic was simulated with the device simulator DESSIS-ISE; the collector voltage was set to 2.0 V and the emitter and substrate contacts were grounded. Ramping the base voltage from

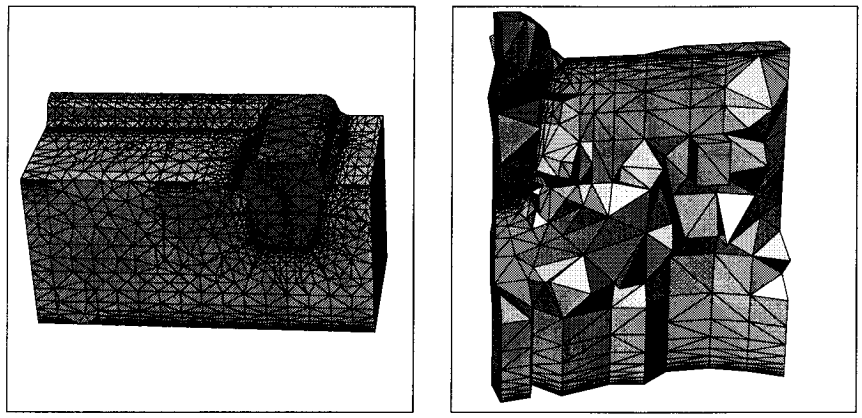


Figure 6.11: *Emitter Coupled Logic transistor; left; surface mesh of final mesh; right: inside view.*

0 V to 1.0 V gives the base and emitter currents shown in Fig. 6.12 in logarithmic scale. A second simulation of this transistor was performed using a mesh generated with an octree approach (using MESH-ISE). The second mesh contains 14197 vertices and 76667 tetrahedra. The simulated collector current for the two meshes are in accordance, but the base currents differ largely for lower base voltages.

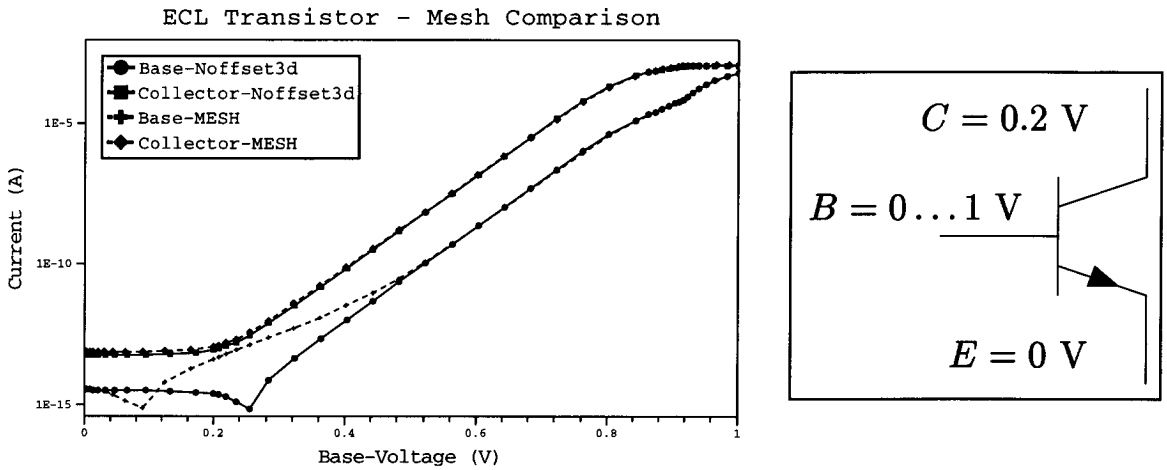


Figure 6.12: *Left: Comparison of simulations of the ECL bipolar transistor, with a normal offsetting mesh and a octree mesh (generated by* MESH-ISE*); right: circuit diagram of the simulation*

3D: Trench isolation

This example demonstrates the extraction of an isosurface and its incorporation into the boundary representation (Module 3D-ISOSURFACE in Sec. 5.7) for a p-n–junction in the shallow trench structure in Fig. 6.13. The data function (i.e. the doping concentration) is defined by analytical functions, which are evaluated on a triangulation. The auxiliary triangulation is computed using MESH-ISE and contains 12730 vertices. By itself this mesh is not fit for a simulation, but it resolves the p-n–junction well. The interpolated patch contains 7164 triangles (Fig. 6.15).

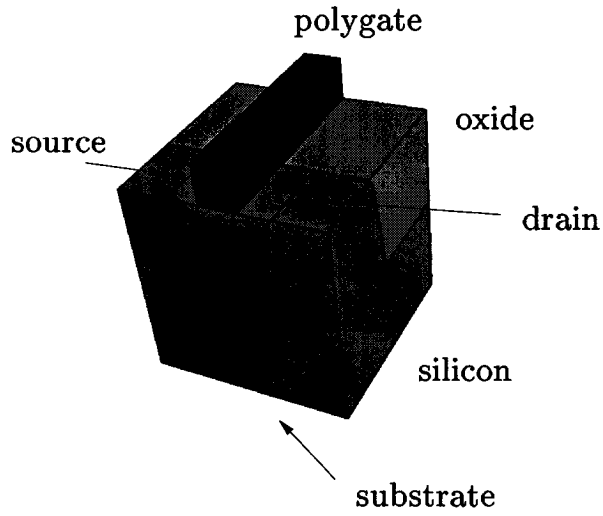


Figure 6.13: *Model*

The re-meshing is done in parametric space and the coarse patches are included into the boundary representation (Fig. 6.15). For the subsequent meshing step the p-n–junction is available like a material interface. The final mesh is displayed in Fig. 6.14 (right) looking from the outside. The mesh contains 15848 points in 93225 tetrahedra. The mesh lines follow the Si/SiO_2 interface and the p-n–junction that was constructed in the previous step.

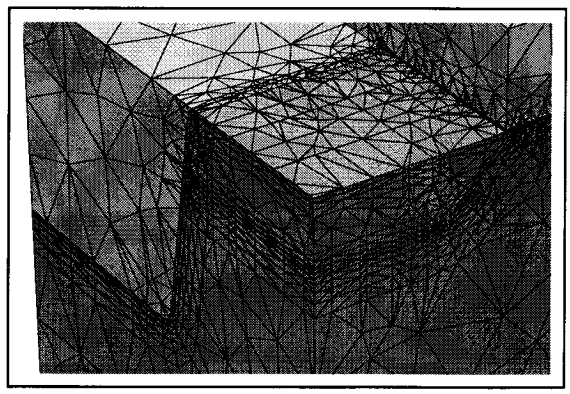


Figure 6.14: *final mesh of this structure*

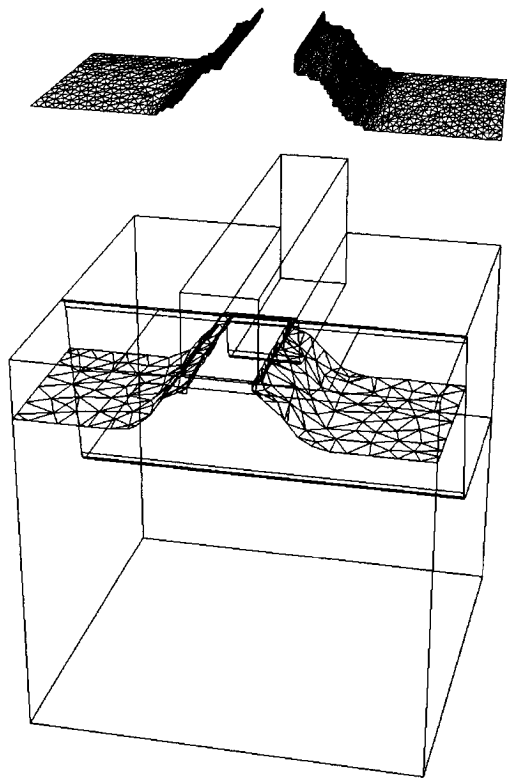


Figure 6.15: *Left: isosurface patches for the junction as extracted from the background mesh and the new boundary representation with junction.*

3D: A surface mesh

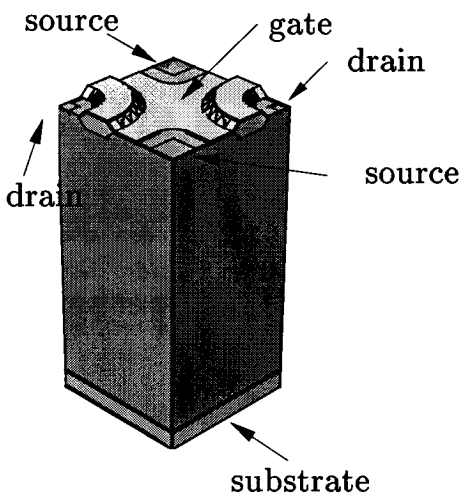


Figure 6.16: *Model of a transistor with a rounded field oxide*

To investigate the influence of the surface mesh, a transistor with a curved field oxide is considered in Fig. 6.16. The simulated silicon region has the form of a brick of size 8 μm $\times$ 8 μm $\times$ 16μm. At the interfaces to the top layers an initial distance of 0.02 μm was chosen for the normal offsetting. Figure 6.17 demonstrate the evolution of the surface mesh. The top view repeats the geometry definition as a close-up at the Si/SiO_2 interface. In the middle, one sees the surface mesh as constructed with the algorithm [25D-NOFFSET]. This surface mesh is adapted to the volume normal offsetting in such a way that the volume mesh can comply with the BMCD criterion. The bottom picture shows the interface of the final mesh after the volume points have been generated and the final Delaunisation has been applied. The main structure of the surface mesh is preserved, but more points have been created, especially around folds and corner points. These points are due to non-Delaunay faces in the volume, i.e. front faces that can be recovered but points in a consecutive layer made these faces non-Delaunay. This example demonstrates that the surface meshing to a large extend creates a suitable surface mesh, but that some non-Delaunay cases are unpredictable.

This effect is due to the conflict of the BMCD criterion and anisotropy for non-planar geometries. In some cases only a lower anisotropy than prescribed can be attained, as the interface mesh becomes finer. The algorithm [25D-NOFFSET] accounts only for conflicts that would arise with any algorithm that creates an anisotropic volume mesh. This examples shows that, in this implementation, an even finer mesh is generated by the Delaunisation.

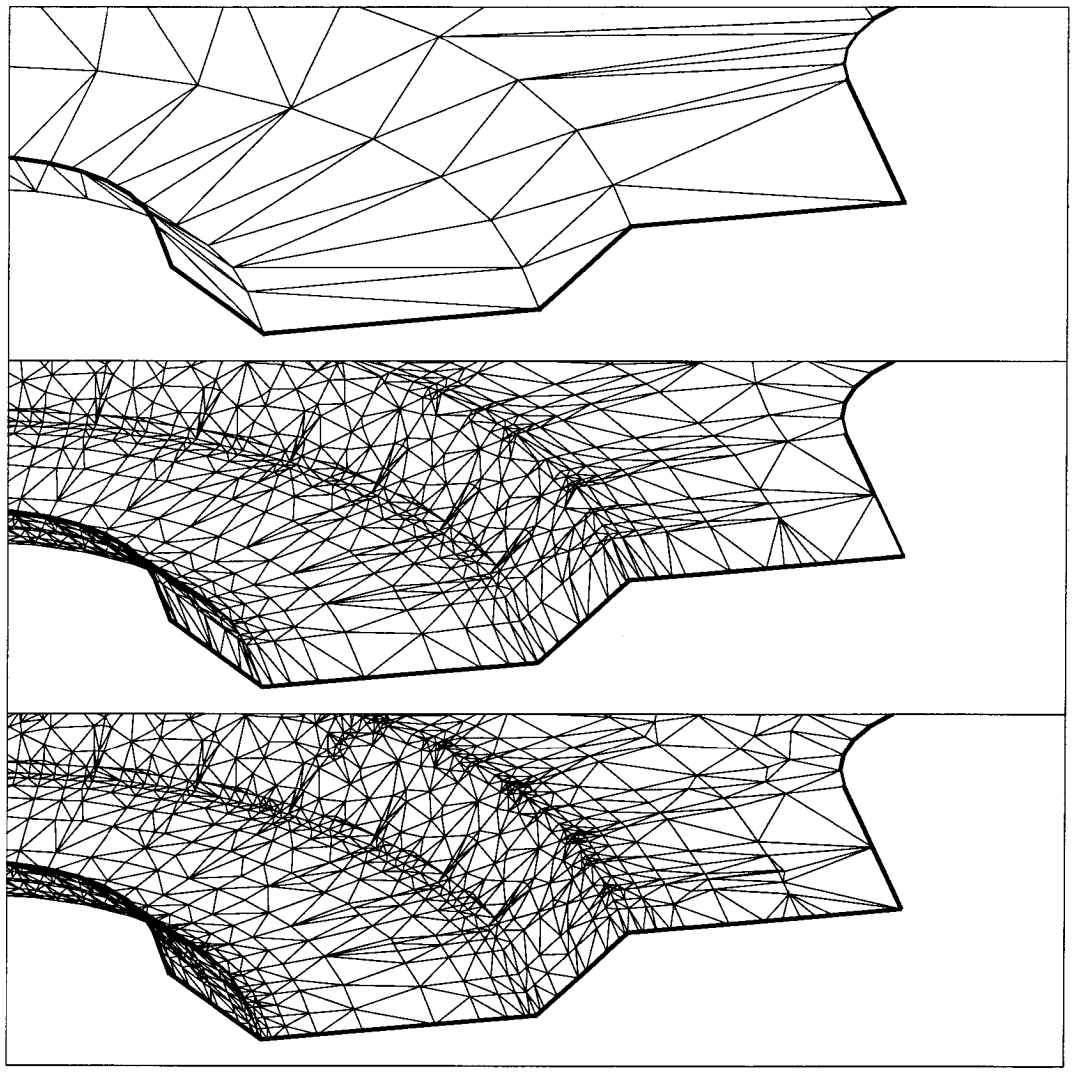


Figure 6.17: *Top: close-up to interface description of the* Si/SiO_2 *interface; middle: surface mesh generated for 3D normal offsetting; bottom: interface mesh of final mesh (with volume points and final Delaunisation).*


Seite Leer /
Blank leaf

Chapter 7

Conclusion and outlook

This chapter looks back at the requirements for the mesh generator and discusses how they are met by the normal offsetting technique. The 3D part, as the most crucial part, is put in the foreground here.

Firstly, the strategy of the reconnection-based normal offsetting is reviewed with the help of the simple 2D example in Fig. 7.1. The input is a simple rectangle and two layers that shall be constructed by the normal offsetting. At first, projections of these layers on the surface are added and in the second step further refinement points are added to the surface. For this surface mesh a volume triangulation is constructed. In the fourth step the points for the normal offsetting are constructed and inserted into the triangulation by a Delaunay technique. Further points are added to the unmeshed void in the fifth phase. The last picture shows that further points can be added in the Delaunisation.

Robustness The critical algorithm for robustness is to generate the empty volume triangulation; after that is constructed a valid triangulation is maintained. Some points may be rejected from insertion, but rejecting a point does not let the entire generator fail. Such a local failure, however, means that some refinement criteria are not met.

Problems to construct a volume triangulation always reduce to an incorrect boundary description, e.g. a volume that is not closed or a

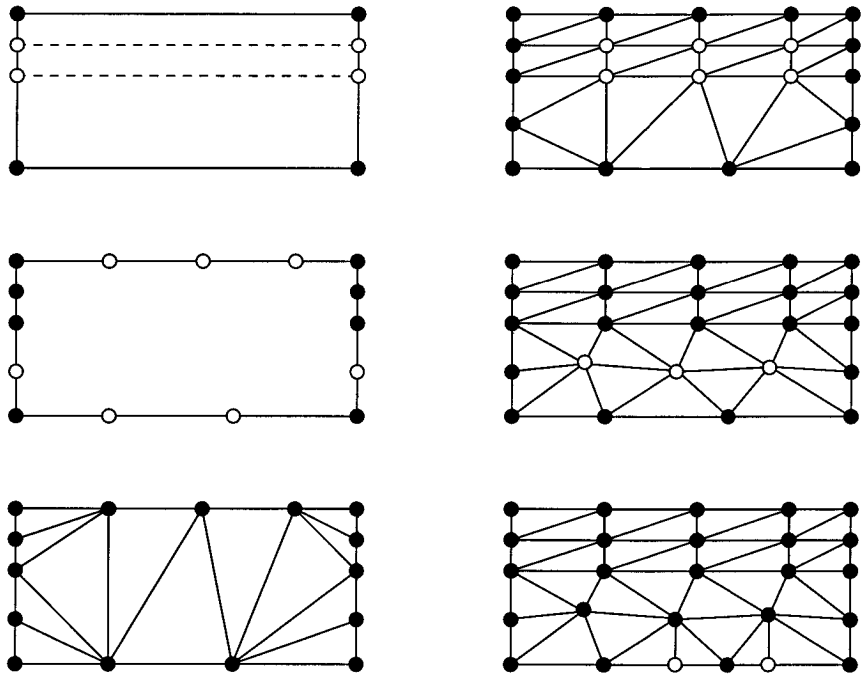


Figure 7.1: *Mesh generation with a reconnection-based normal offsetting with a simple 2D example. The points added at each step are filled white.*

non-conformal description.

Delaunay condition The BMCD triangulation using an incremental Delaunay construction can create strictly Delaunay meshes if exact predicates are used. It may create sliver elements, especially with points created by the normal offsetting algorithm, but these can be suppressed by the discussed method with the cost of (slightly) non-Delaunay meshes.

Anisotropic layers Normal offsetting together with its surface meshing algorithm can create layers with anisotropic elements along interfaces. The limits to anisotropy at non-planar regions of the model are mainly due to the BMCD criterion and therefore method independent. A different approach would have the same limits. Normal offsetting surface meshing tries to find an optimal input mesh for the volume layering. The anisotropy can, however, be further lowered by conflicts in the volume meshing. If faces and edges have to be recovered by transformations

(because they are not Delaunay) they induce further point insertions in Delaunisation module. In the case when several anisotropic layers are affected one single forced face can lead to a series of inserted points.

User requirements Apart from the normal offsetting construction, the volume mesh needs further refinement. These points are created by a simple bisection algorithm. Other approaches, even external sources, can be easily plugged into the modular structure of the software design.

Application Since mesh generation is part of a simulation environment its result must be applied there. The 2D version is part of commercial products: it is one of the algorithms in the mesh generator MESH-ISE for device simulation and as an alternative meshing engine in the process simulation DIOS-ISE. In 3D, first tests for device simulation have been performed.

For the use in the 3D process simulator further algorithms need to added:

- As the geometry by itself is part of the simulation, a new model description has to be computed several times during the simulation. The research for such a moving boundary algorithm for process simulation has not yet reached the necessary robustness [KSVF00a].
- After the mesh for a new time step is generated the data function must be interpolated on the new mesh. In order to minimise the interpolation error it is preferable to keep as many points as possible at the same location (shifted by the computed displacement). A concept would be to start normal offsetting from the new interface and use the points from the old mesh as an external point source for the field points. The transition between the layers and the volume points may need further refinement.
- The treatment of thin layer regions is not yet satisfactory; the 2D method certainly cannot be lifted to 3D. The difficulty is the simultaneous control of the refinement points in several layers.

Bibliography

[BE92] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, volume 1, pages 23–90. World Scientific, 1992.

[BF97] H. Borouchaki and P. Frey. Maillage géometrique de surface I: enrichissement, II: apauvirissement. Rapport de Recherche RR-3236, INRIA, 1997.

[BG97] H. Borouchaki and P.-L. George. Aspects of 2D mesh generation. *International Journal for Numerical Methods in Engineering*, 40(11):1957–1975, 1997.

[Bow81] A. Bowyer. Computing Dirichlet tesselations. *The Computer Journal*, 24(2):162–166, 1981.

[BP91] J. Bonet and J. Peraie. An alteranting digital tree (ADT) for 3D geometric and intersection problems. *International Journal for Numerical Methods in Engineering*, 31:1–17, 1991.

[BRF83] R. E. Banks, D. J. Rose, and W. Fichtner. Numerical methods for semiconductor device simulations. *IEEE Transactions on Electron Devices*, 30(9):1031–1041, 1983.

[FBG96] P. J. Frey, H. Borouchaki, and P.-L. George. Delaunay tetrahedralization using an advancing–front approach. *Proceedings of 5th International Meshing Roundtable*, pages 31–46, 1996.

[Fle99] P. Fleischmann. *Mesh Generation for Technology CAD in Three Dimensions.* Ph.D. thesis, TU Vienna, Austria, December 1999.

[For92] S. Fortune. Voronoi diagrams and Delaunay triangulation. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, volume 1, pages 193–233. World Scientific, 1992.

[FRB83] W. Fichtner, D. J. Rose, and R. E. Bank. Semiconductor device simulation. *IEEE Transactions on Electron Devices*, 30(9):1018–1030, 1983.

[Gar99] G. Garretòn. *A Hybrid Approach to 2D and 3D Mesh Generation for Semiconductor Device Generation.* Ph.D. thesis, ETH Zurich, Integrated Systems Laboratory, Switzerland, 1999.

[GB98] P.-L. George and H. Borouchaki. *Delaunay Triangulation and Meshing, Application to Finite Elements.* Editions Hermes, Paris, 1998.

[GH99] P.-L. George and F. Hecht. Nonisotropic grids. In J. F. Thompson, B. Soni, and N. Wheatherhill, editors, *Handbook of Grid Generation*, chapter 20. CRC Press, 1999.

[GHS88] P.-L. George, F. Hecht, and E. Saltel. Tétraédrisation automatique et respect de la frontière. Rapport de Recherche RR-835, INRIA, 1988.

[GHS91] P.-L. George, F. Hecht, and E. Saltel. Automatic mesh generator with specified boundary. *Computer Methods in Applid Mechanics and Engineering*, 92:269–288, 1991.

[HR00] N. Hitschfeld and M.-C. Rivara. Quality nonobtuse boundary an/or interface Delaunay triangulations. In B. Soni et al, editor, *Proceegings of 7th International Conference on Numerical Grid Generation in Computational Field Simultions*, pages 285–294. ISGG, 2000.

[HS88] D. G. Holmes and D. D. Snyder. The generation of unstructured triangular meshes using Delaunay triangulation. *Numerical Grid Generation in Computational Fluid Mechanics 1988*, pages 643–652, 1988.

[Int99a] Integrated Systems Engineering AG, Zurich, Switzerland. *Dessis-ISE*, 6.0 edition, 1999.

[Int99b] Integrated Systems Engineering AG, Zurich, Switzerland. *Dios-ISE*, 6.0 edition, 1999.

[Int99c] Integrated Systems Engineering AG, Zurich, Switzerland. *Mesh-ISE*, 6.0 edition, 1999.

[JS92] B. P. Johnston and J. M. Sullivan. Fully automatic two dimensional mesh generation using Normal Offsetting. *Proceedings of International Journal for Numerical Methods in Engineering*, 33:425 – 442, 1992.

[KKM95] Y. Kallinderis, A. Khawaja, and H. McMorris. Hybrid prismatic/tetrahedral grid generation for complex geometries. Technical Report 95-0211, AIAA, 1995.

[KSF00] J. Krause, N. Strecker, and W. Fichtner. Boundary-sensitive mesh generation using an offsetting technique. *International Journal for Numerical Methods in Engineering*, 49:51 – 59, 2000.

[KSVF00a] J. Krause, B. Schmithüsen, L. Villablanca, and W. Fichtner. New developments and old problems in grid generation and adaptation for TCAD applications. *IEICE Transactions*, E83-C(8):1331–1337, 2000.

[KSVF00b] J. Krause, N. Strecker, L. Villablanca, and W. Fichtner. Robust anisotropic 3D grid generation using a normal offsetting approach. In B. Soni et al., editor, *7th International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 305–314. ISGG, 2000.

[KW98] L. Kettner and E. Welzl. One-sided error predicates in geometric computing. *Proceedings of the XV. IFIP World Computer Congress*, pages 13 –25, 1998.

[Law72] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3:365–372, 1972.

[Löh96] R. Löhner. Extensions and improvements of the AFT grid generation techniques. *Communications in Numerical Methods in Engineering*, 12:683 – 702, 1996.

[LP88] R. Löhner and P. Parikh. Generation of 3D unstructred grids by the advaning front method. *International Journal for Numerical Methods in Engineering*, 8:1135–1149, 1988.

[MH95] P. Möller and P. Hansbo. On advancing front mesh generation in 3d. *International Journal for Numerical Methods in Engineering*, 38:3551 – 3569, 1995.

[MRS90] P. S. Markowich, C. A. Ringhofer, and C. Schmeiser. *Semiconductor Equations.* Springer-Verlag, Wien, New York, 1990.

[MTT+96] G. L. Miller, D. Talmor, S.-H. Teng, N. Walkington, and H. Wang. Control volume meshes using sphere packing: Generation, refinement, and coarsening. In *Proceedings of 5th International Meshing Roundtable*, pages 47–61, 1996.

[Mül94] S. Müller. *An object-oriented approach to multidimensional semiconductor device simulation.* Ph.D. thesis, ETH Zurich, Integrated Systems Laboratory, Switzerland, 1994.

[MW95] D. L. Marcum and N. P. Weatherill. Unstructured grid gerneration using iterative point insertion an local reconnection. *AIAA Journal*, 33:1619–1625, 1995.

[Péb98] P. P. Pébay. Construction d'une triangulation surfacique Delaunay-admissible. Rapport de Recherche RR-3369, Inria, 1998.

[PK96] V. Parthasarathy and Y. Kallinderis. Adaptive prismatic-tetrahedral grid refinement and redistribution for viscous flows. *AIAA Journal*, 34(4):707–716, 1996.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[PZSF00] A. Pomp, S. Zelenka, N. Strecker, and W. Fichtner. Viscoelastic material behavior: Models and discretization in process simulator DIOS. *IEEE Transactions on Electron Devices*, 47(10):1999–2007, 2000.

[RS92] J. Ruppert and R. Seidel. On the difficulty of triangulating 3D nonconvex polyhedra. *Discrete and Computational Geometry*, 7:227–253, 1992.

[Rup95] J. Ruppert. A Delaunay refinement algorithm for quality 2D mesh generation. *Journal of Algorithms*, 18:548–585, 1995.

[Sch97] J. Schöberl. NETGEN: An advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997.

[SdS00] A. Sheffer and E. de Sturler. Surface parameterization for meshing by triangulation flattening. In *Proceedings of 9th International Mesching Roundtable*, pages 161–171. Sandia National Laboratories, 2000.

[Sev97] E. Seveno. Towards an adaptive advancing front method. *Proceedings 6th International Meshing Roundtable*, pages 349–360, 1997.

[SG69] D. L. Scharfetter and H. K. Gummel. Large-signal analysis of a silicon Read diode oscillator. *IEEE Transaction on Electron Devices*, ED-16(1):64–77, 1969.

[She97a] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, 18:305–363, 1997.

[She97b] J. R. Shewchuk. *Delaunay Refinement Mesh Generation.* Ph.D. thesis, Carnegie Mellon University, Pittsburgh, 1997.

[Str97] B. Stroustrup. *C++ Programming Language.* Addison-Wesley, 3rd edition, 1997.

[Suz90] Masahiro Suzuki. Surface grid generation on unstructured grids. *AIAA Journal*, 12:2263–2264, 1990.

[Vil00] L. Villablanca. *Mesh Generation Algorithms for 3D Semicondctor Process Simulation.* Ph.D. thesis, ETH Zurich, Integrated Systems Laboratory, Switzerland, 2000.

[Wat81] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytops. *The Computer Journal*, 24(2):167–172, 1981.

[WM96] D. J. Walton and D. S. Meek. A triangular G^1 patch from boundary curves. *Computer Aided Design*, 28(2):113–123, 1996.

Appendix A

Glossary

AFT Advancing Front Technique

BMCDT Box Method Conforming Delaunay Triangulation

circumsphere the sphere that passes through all points of a tetrahedron.

diametral sphere The smallest sphere that contains the two points of an edge; the centre is identical with the midpoint of the edge.

DT Delaunay Triangulation

equatorial sphere The smallest sphere that contains the three points of a triangle or rectangle; the centre of the sphere is identical with the centre of the circumcircle of the triangle/rectangle in the plane defined by the face.

FEM Finite Element Method

grid An structured covering of the domain with axis aligned elements like rectangles and bricks.

mesh An unstructured covering of the domain with certain types of elements, suitable for numerical calculations. Allowed elements depend on the application; in 2D e.g. triangles, rectangles, or quadrilaterals, whereas 3D has tetrahedrons, bricks, prisms, hexahedra, and others.

medial axis set of points, that have the same distance to two different points of a polygon.

Steiner Point Volume refinement point in Delaunay methods, usually at Voronoï centres.

Appendix B

Command File Description

The practical implementation of this work is driven by a command file. The grammar of this language is defined and the connection to the particular algorithms is pointed out. In that way this appendix serves as a reference manual.

How to read this

Words in these grammar tables have three different meanings, and are typed accordingly:

`typewriter:` terminal keywords

italics: nonterminal symbols, that are resolved later

times roman: nonterminal symbols, that are not explained, but are self-explanatory, like 'float', 'int', 'double-quoted-string' etc.

Some characters act as word separators (`\t, \n, " "`) and are not included in the description.

File

file:
  `title` double-quoted-string *blocks*
blocks:
  blocks block
  block
block:
  ignore-block-keyword {anything-containing-balanced-braces }
  `Offsetting` {*offsetting-block* }
ignore-block-keyword:
  `Control`
  `Definitions`
  `Placements`

The structure of the command file for MESH-ISE is enriched by a by one block. The other blocks are ignored; except for versions linked with DLIB-ISE: those versions can use data functions defined in the command file for the generator MESH-ISE (see [Int99c]).

Offsetting block

offsetting-block:
  offsetting-block-lines
offsetting-block-lines:
  offsetting-block-lines offsetting-block-line
  offsetting-block-line
offsetting-block-line:
  `usebox` = *bool*
  `maxangle` = float
  `background` = double-quoted-string
  `backgrounddata` = double-quoted-string
  `options` = double-quoted-string
  noffset-block
  boundary-block
  thinlayer-block

From the top-level block global algorithmic switches can be used, and

several subsections can be accessed.

name	meaning	default
`usebox`	with link to DLIB-ISE: use refinement boxes	0
`maxangle`	2D: suppress large angles	180.0
`background`	auxiliary grid	""
`backgrounddata`	and data defined on it	""
`options`	discussed below	

The value for `maxangle` is used in 2D/2.5D-REFINE to suppress large angles. Note that a value of smaller than 120^o can result in infinite loops.

Versions linked with DLIB-ISE refine (in 2D/2.5D/3D-REFINE) the mesh using the information for 'Refinement' in the language for MESH-ISE [Int99c] if `usebox` is set.

The mesh and data files in `background` and `backgrounddata` are loaded in the Module 3D-ISOSURFACE as data source for the isosurface computations. The `options` (see below) select the series of algorithms applied in the 3D generator.

Noffset

noffset-block:
  noffset {*noffset-lines* }
  noffset *region-or-material* double-quoted-string \
    double-quoted-string {*noffset-interface-lines* }
  noffset *region-or-material* double-quoted-string \
    {*noffset-region-lines* }
noffset-lines:
  noffset-lines noffset-line
  noffset-line
noffset-line:
  noffset-interface-line
  noffset-region-line

noffset-interface-lines:
noffset-interface-lines noffset-interface-line
noffset-interface-line
noffset-region-lines:
noffset-region-lines noffset-region-line
noffset-region-line
noffset-interface-line:
`hloc` = float
`factor` = float
`subdivide` = int
noffset-region-line:
`maxedgelength` = float
`maxlevel` = int
`terminateline` = int

There are three types of blocks; firstly for defining the global default, secondly specifying interface parameters, and specifying region parameters. The latter two can be defined using region or material names, with the appropriate keyword. All parameters have global defaults, some can be specified per region other per interface.

`name`	meaning	default
`hloc`	thickness of first layer	0.1
`factor`	coarsening factor	1.3
`subdivide`	number of subdivisions	0
`maxedgelength`	maximum edge length allowed in volume	max-float
`terminateline`	number of rigid layers	3
`maxlevel`	maximum number of layers	200

The main parameters for normal offsetting (2D/3D-NOFFSET) are `hloc`, `factor`, and `maxlevel`. They decide on the thickness of the first layer, the coarsening and the number of layers created.

The number `terminateline` controls for how many layers the algorithm 2D-TERMINATELINE (p.30) is *not* called, so that, locally, a warped tensor grid is created.

The parameter `maxedgelength` is observed by all refining algorithms.

Surface mesh

boundary-block:
  boundary {*boundary-lines* }
  boundary *region-or-material* double-quoted-string \
    double-quoted-string {*boundary-interface-line* }
boundary-lines:
  boundary-lines boundary–line
  boundary-line
boundary-line:
  boundary-interface-line
  balance = *bool*
  refine = *bool*
boundary-interface-line:
  hglob = float
isoline-block:
  isoline double-quoted-string {*isoline-lines* }

The command file features two types of blocks for boundary grid; firstly for defining the global default, secondly specifying interface parameters. The latter can be defined using region or material names, with the appropriate keyword. All parameters have defaults, some can be specified per interface. The algorithm switches have only a global defaults. A second block defined in the same or overlapping scope takes precedence.

name	meaning	default
balance	balancing of 1D grid	1
refine	refine corners more	1
hglob	discretisation length	1.0

The value **hglob** (if it is smaller than **maxedgelength** for the neighbouring regions) is applied in the boundary meshing algorithms 2D/3D-BOUNDARY. **Balance** gives the possibility to switch on mesh balancing for 2D-BOUNDARY (p. 35).

The parameter **refine** controls the 2D/3D-REFINE-CURVATURE algorithms.

isoline-lines:
isoline-lines isoline-line
isoline-line
isoline-line:
species = double-quoted-string
value = float
region-or-material = double-quoted-string
submesh = double-quoted-string

For version that a linked together with DLIB-ISE the isoline statement has effect, also the ignored sections are read by that library and can be referenced. The isoline (isosurface in 3D) computation takes a given name as identity specification. Note that the default values define the p-n–junction in silicon. In the 2D implementation, the **submesh** statement defines as data source the submesh statement in the placement section of this command file. If is equal to the empty string, a triangulation is computed on the fly and function definitions of this command file are evaluated on this trinangulation.

name	meaning	default
species	dataset to evaluate	"DopingConcentration"
value	level of isoline	0.0
region-or-material	region or material where to insert line	material "Silicon"
submesh	data source	""

thinlayer-block:
thinlayer double-quoted-string {*thinlayer-lines* }
thinlayer-lines:
thinlayer-lines thinlayer-line
thinlayer-line
thinlayer-line:
region = {double-quoted-string double-quoted-string \
double-quoted-string}
thickness = float
deviationdist = float
deviationportion = float
angle = float

The `region` keyword specifies the three layers, that define a thin layer if it is thinner than `thickness`. The parameters `deviationdist` and `deviationportion` control, how much the geometry is allowed to change (in absolute and in relative lengths) The parameters have no reasonable default:

name	meaning
`region`	three region name defining the sandwich
`thickness`	only thinner portion are treated
`deviationdist`	distance an interface may change in absolute value
`deviationportion`	relative of layer thickness
`angle`	angle tolerance

Other terminal symbols

region-or-material:
  `region`
  `material`
bool:
  `0`
  `1`

Phases

The `options` keyword can control the series of algorithms for the 3D generator. The keywords correspond to the algorithms in the following table.

`isosurface`	3D-ISOSURFACE
`surface`	3D-BOUNDARY
`noffset3d`	3D-NOFFSET
`deltri`	3D-TRIANGULATION, 3D-BMCDT
`refine`	3D-REFINE

For the complete mesh generation the default is a good choice: "`-p surface -p deltri -p noffset3d -p refine -p deltri`". If only sin-

gle algorithms or the isosurface capability are studied different settings can be used.

Appendix C

A generic implementation of a Delaunisation algorithm

Modern programming languages give multiple possibilities to reuse code or to use the same code in different contexts. This work offers an opportunity to employ these features. The local mesh optimisation algorithm described in Sec. 5.3 can be applied on different mesh data structures: in the 2D plane, for surfaces in 3D space, and when 3D surface patches are treated in a parametric space. It would be tedious to write the same code three times for three different mesh data structures! The language C++ [Str97] offers the *template* construct to implement generic algorithms, in this case the code needs to be written once. Also maintenance becomes simpler.

A *template* is parameterised variable type (or class) in the definition of a function or a class. The compiler replaces the template name by the type name for which an instance is called. Only types are allowed for which the operations and function are defined that are used in the template code. Failure to do so results in a compiler errors, hence there is no runtime penalty in using templates because the code for different instances is entirely generated by the compiler and can be optimised.

In general the mesh data structures do not offer the same interface to algorithms in such a strict sense that the template mechanism can find the equivalences: the compiler needs exactly the same names for overloaded functions. Therefore, an interface class is used as template parameter (`MeshInterface`) in the optimiser algorithm, which is implemented in a class `Optimiser`. The data structure handling is, by virtue of these two classes, split into an algorithm specific part and in a data structure dependent part. The functions in the interface class `MeshInterface` are in most cases wrappers for functions already existing in the underlying data structure. These wrappers can be implemented as one-liners with the *inline* qualifier, which instructs the compiler not create a function but to replace the invocation by this code segment. This optimisation removes the overhead by an additional function call.

Only the basic functionality of the class layout for `Optimiser` is listed below, for example the protection sphere mechanism is not mentioned. The geometric operators (`SwapEdge`, `RefineEdge`) and the queries `RefinementPoint` and `IsSurfaceDelaunay` are shown here.

```
template<class MeshInterface>
class Optimiser{
 //structure for mesh data
 MeshInterface::Mesh   * mesh;
 //stack with edges to be checked
 Stack<Edge> stack;
 //find best refinement point (inter-sector/edge mid)
 void RefinementPoint(const Edge &e,
                      MeshInterface::Vector &refinevec);
 //in-sphere-test
 bool IsSurfaceDelaunay(const Edge &e);
 //geometric operators
 bool SwapEdge(Edge &e);
 void RefineEdge(Edge &e, const MeshInterface::Vector &v);
public:
 //constructor and destructor
 Optimiser(MeshInterface::Mesh *);
 ~Optimiser();
 //improves mesh locally, starting with edges in stack
 void OptimiserLocally();
 //improves mesh globally
```

```
 void OptimiserGlobally();
 //add edge to the stack
 void PushEdge(Edge &e);
};
```

The interface class `MeshInterface` has to be defined for each mesh data structure that uses the `Optimiser` algorithm. This listing only names the type and functions needed, it does not implement a specific interface. Firstly it forwards the class names for the entities that constitute the mesh and the vector class type. These types can be primitive type, like in this implementation the entities are referred to as indexes in an array. The wrapper function queries the topological and geometrical information of the underlying data structure. Also creation and removal of those entities is included. Again, only a simplified layout is shown, e.g. the handling of material regions is neglected.

```
class MeshInterfaceXD{
public:
 //types
 typedef XDMesh   Mesh;      //structure for mesh data
 typedef XDPoint  Point;     //point class
 typedef XDEdge   Edge ;     //edge class
 typedef XDFace   Face;      //face class
 typedef XDVector Vector;    //vector class
 //is geometric feature, or constraint ?
 static bool IsFold(Mesh * m, const Edge &e);
 static bool IsConstraint(Mesh * m, const Edge &e);
 //set constraint
 static bool Constrain(Mesh * m, const Edge &e);
 //how many faces are attached to this edge
 static INTEGER GetNumFaces(Mesh * m, const Edge &e);
 //get i'th face at edge e
 static Face GetFace(Mesh * m, const Edge &e, int i);
 //get points in edge
 static Point GetEdgePoint(Mesh * m, const Edge &e, int i);
 //is edge existing
 static bool IsValidEdge(Mesh * m, const Edge &e);
 //find point in f opposite to e
 static Point GetOtherPointFace(Mesh * m, const Face &f,
```

```
                                    const Edge &e);
 //find egde in hash table or neighbourhood search
 Edge FindEdge(Mesh * m, const Point &p, const Point &q);
 //remove
 static void RemoveEdge(Mesh * m, Edge &e);
 static void RemoveFace(Mesh * m, Face &f);
 //create
 static Edge  CreateEdge(Mesh * m, Point &p, Point &q);
 static Face  CreateTriangle(Mesh * m, Point * p);
 static Point CreatePoint(Mesh * m, const Vector &v);
 //get vector
 static Vector GetVector(Mesh * m,const Point &p);
};
```

The code for the main loop of the optimisation retrieves an edge from the stack and tests, by visiting adjacent triangles, if it is optimal (`IsSurfaceDelaunay`). If not, the edge is either swapped (`SwapEdge`) or refined (`RefineEdge`). The code only uses functions from the `Optimiser` class and the abstract interface, i.e. direct calls to the underlying data structure are avoided.

```
template<class MeshInterface>
void Optimiser<MeshInterface>::OptimiserLocally(){
  Edge e;
  while(stack.Pop(e))
    //if edge already removed
    if(MeshInterface::IsValidEdge(mesh,e)){
      bool swapable = !MeshInterface::IsFold(mesh,e)
            && !MeshInterface::IsConstraint(mesh,e);
      if(!swapable) continue;
      //BMCD in-sphere-test
      bool isdel = IsSurfaceDelaunay(e);
      if(isdel) continue;
      if(swapable)
        SwapEdge(e);
      else{
        MeshInterface::Vector refine_point;
        RefinementPoint(e,refine_point);
        RefineEdge(e,refine_point);
```

```
      }
    }
}
```

As an example of the generic implementation of the operators the `RefineEdge` function is listed. Here the underlying data structure is accessed via the abstract interface. The possible multidimensionality of the mesh is accounted for in the fact that an edge may have more than two neighbouring faces.

```
template<class MeshInterface>
void Optimiser<MeshInterface>::RefineEdge(Edge &e,
                  const MeshInterface::Vector &v){
  Point p[] = {MeshInterface::GetEdgePoint(mesh,e,0),
               MeshInterface::GetEdgePoint(mesh,e,1)};
  Point point = CreatePoint(v);
  Edge x;
  bool isconstraint = MeshInterface::IsConstraint(mesh,e);
  //remove faces and recreate children
  while(MeshInterface::GetNumFaces(mesh,e) != 0){
    Face f = MeshInterface::GetFace(mesh,e,0);
    Point op = MeshInterface::GetOtherPointFace(mesh,f,e);
    MeshInterface::RemoveFace(mesh,f);
    Point pp[]={p[0],point,op,p[1]};
    MeshInterface::CreateTriangle(mesh,pp);
    MeshInterface::CreateTriangle(mesh,pp+1);
    //update stack
    x = MeshInterface::FindEdge(mesh,op,p[0]);
    stack.Push(x);
    x = MeshInterface::FindEdge(mesh,op,p[1]);
    stack.Push(x);
  }//for faces at 'e'
  //update stack, inherit constraint
  x = MeshInterface::FindEdge(mesh,p[0],point);
  stack.Push(x);
  if(isconstraint)
    MeshInterface::Constrain(mesh,x);
  x = MeshInterface::FindEdge(mesh,p[1],point);
  stack.Push(x);
```

```
  if(isconstraint)
    MeshInterface::Constrain(mesh,x);
  MeshInterface::RemoveEdge(mesh,e);
}
```

Curriculum Vitæ

Jens Krause was born in Rheinbach/Germany on November 26th, 1968. After visiting grammar school in Euskirchen, he fulfilled the community service in the hospital of Mechernich. He studied physics at the Rheinische-Friedrich-Wilhelms Universität in Bonn/Germany and at the University of New South Wales in Sydney, Australia. He received the degree of "Diplomphysiker" in 1996 of Bonn University. From 1996 to 1997 he work as a software engineer in Hennef/Sieg in Germany. In 1997 he joined the Integrated Systems Laboratory of the Swiss Federal Institute for Technology (ETHZ). His main research interest is mesh generation for semiconductor process and device simulation.