Applied Mathematics and Computation 267 (2015) 195–206

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Transformation invariant local element size specification

Florian Rudolf [a,*], Karl Rupp [a,b], Josef Weinbub [a], Andreas Morhammer [c], Siegfried Selberherr [a]

[a] Institute for Microelectronics, TU Wien, Gußhausstraße 27-29/E360, 1040 Wien, Austria
[b] Institute for Analysis and Scientific Computing, TU Wien, Wiedner Hauptstraße 8-10/E101, 1040 Wien, Austria
[c] Christian Doppler Laboratory for Reliability Issues in Microelectronics, Gußhausstraße 27-29/E360, 1040 Wien, Austria

ARTICLE INFO

Keywords:
Mesh generation
Element sizing
Mesh refinement
ViennaMesh

ABSTRACT

Quality and size of mesh elements are important for optimizing the accuracy and convergence of mesh-based simulation processes. Often, a priori information, like internal material properties, of regions of interest is available, which can be used to locally specify the mesh element size for finding a good balance between the mesh resolution on the one hand and the runtime and memory performance on the other. In many applications, like the optimization of geometric parameters, multiple meshes of similar objects are required. Typical mesh element size specification methods, like scalar fields, are inflexible because of their dependence on the geometry of the object. To avoid the creation of a mesh element size specifications for each object manually, a specification method based on the objects topology rather than on its geometry, is needed. We tackle this problem by extending our meshing software ViennaMesh with a dynamic framework for locally specifying the size of mesh elements. Our approach aims for convenient utilization by using a XML-based configuration with support for arithmetic expressions. To achieve a high level of flexibility and reusability, this configuration can be specified based on the object's topology, for example interfaces between different material regions. Additionally, geometric parameters, like the radius of the circumsphere of the object, are provided and can be used to, e.g., scale the local mesh element size according to the total size of the object. As a result, our configuration method is invariant under large set transformations, especially deformations, of the object enabling a high level of geometry independence. We depict the practicability of our approach by providing examples for meshes generated with this element sizing framework and discussing a geometry optimization application.

1. Introduction

Methods like the finite element method (FEM) or the finite volume method are of widespread use for simulation processes governed by partial differential equations. Examples include technology computer-aided design [1] or computational fluid dynamics [2]. These methods require the simulation domain, we call it Ω, to be characterized by a *geometry*, which defines its shape and size. Using this geometry, Ω is discretized into a finite number of primitive geometric elements. Without going into the full mathematical details, the resulting discretization is called *mesh* and the discretization process is referred to as mesh generation [3,4]. Popular mesh elements are triangles and quadrilaterals as well as tetrahedrons or

* Corresponding author.
E-mail addresses: rudolf@iue.tuwien.ac.at (F. Rudolf), rupp@iue.tuwien.ac.at (K. Rupp), weinbub@iue.tuwien.ac.at (J. Weinbub), morhammer@iue.tuwien.ac.at (A. Morhammer), selberherr@iue.tuwien.ac.at (S. Selberherr).

http://dx.doi.org/10.1016/j.amc.2015.04.027
0096-3003/

hexahedrons for domains of two and three dimensions, respectively. A simulation domain can also be partitioned into different regions, for example material regions. Interfaces between material regions may induce additional interface conditions which might need special consideration and are easier to handle if resolved by the mesh. In this work, such a partition is referred to as *mesh region*.

The shape and size of the mesh elements directly affect the convergence behavior of simulations [5]. Let u be the true solution with sufficient regularity and u_h be the discrete solution for a mesh with characteristic mesh element size h, then standard error indicators for the various discretization methods are of the form $||u - u_h|| < h * C(u)$, where $C(u)$ is a constant coefficient depending on u and possibly other auxiliary quantities such as the minimum interior angle of all elements in the mesh. In this work we focus on the mesh element size h, assuming that elements are of proper shape. A smaller mesh element size usually leads to a smaller discretization error but results in a higher number of elements, which increases both memory consumption and runtime. A balance between the numerical quality and the memory consumption as well as the runtime has to be found.

Due to modeling, geometry, or boundary conditions, special regions commonly exist where a fine mesh element resolution has a much higher positive impact on the accuracy of the simulation result than other regions. Therefore, the element sizes within a mesh must be non-uniform to achieve the best possible performance in runtime and memory consumption. As known from hp-FEM, mesh element size reduction (h-refinement) in regions with or close to singularities is more efficient than increasing the polynomial order of the finite element basis function (p-refinement) [6], making these regions good candidates for a fine mesh element resolution. In many engineering problems these regions are known a priori and can be specified as input parameters to the mesh generation process. For example, when performing a simulation of a transistor device, the area near the gate requires a fine mesh element resolution while in the bulk region larger elements are sufficient.

In many applications, like a geometry optimization process of a semiconductor device, multiple simulations of different objects are performed [7]. The geometries of these objects often are very similar and only differ in specific regions or geometric features, like the gate length of a transistor device. Therefore, a mesh element size specification mechanism which is able to handle different similar geometries is advantageous.

Most available mesh generation software packages support specification of mesh element sizes [8–11]. There are three popular methods on how to specify mesh element sizes: a global value, a (discrete) scalar field, and a callback function.

A global value defines an upper size bound for all elements in a mesh. While a global value is very convenient and independent of the geometry, it lacks flexibility and locality. Some mesh generation software packages allow specifying different mesh element sizes for different mesh regions, adding a bit of flexibility and locality.

A (discrete) scalar field is a set of element size values and positions in the simulation domain indicating a local environment where the element size value is valid. Using an interpolation technique, an arbitrary position within the simulation domain can be mapped to the local mesh element size. Scalar fields are more flexible than the specification of global values, but there are two disadvantages: First, they are only valid for a specific geometry. Second, the usage of multiple scalar fields simultaneously is non-trivial, because the points, for which the mesh element sizes are specified, might not match and additional interpolation is required.

Callback functions are functions written in a programming language which are used by the mesh generation software to determine local mesh element sizes. Usually, a callback function can be implemented outside the software package without touching the software's source code. Depending on the design of the mesh generation software, the callback function either returns the local mesh element size based on a location within the simulation domain (location-based callback function) or determines, if a certain mesh element is too large and has to be refined (element-based callback function).

Conceptually, the first is similar to a scalar field, where also a location is mapped to a mesh element size, but no interpolation is needed. This mechanism offers a very high level of flexibility but is not convenient to use, because in most cases a re-compilation is required. Additionally, typical interfaces of callback functions just provide local information, such as the position, but no global context, like the mesh region, in which the position is included. Therefore, a callback function which handles global context evaluation has to be provided. Depending on the implementation of the callback function, this mechanism can be independent of the geometry.

Fig. 1 visualizes examples of two meshes generated by using a global value and a scalar field mesh size, respectively. Most mesh generation software supports some of these popular mesh element specification methods, but often these specification methods are incompatible. For example, the file formats for scalar fields are different in software packages. Another major incompatibility is the different interpretation of the size of a mesh element. While some software packages define the size of a mesh element as the size of the longest edge, other software packages use the volume of the mesh element to define its size. These incompatibilities impedes the usage of multiple different mesh generation algorithms with the same mesh element size specification, hence a unified approach is desirable.

ViennaMesh [12] addresses these problems with its *element sizing framework*, a mechanism for specifying local mesh element sizes in a uniform and compatible manner, enabling interchangeability of meshing software. This element sizing framework enables local mesh element size specifications which are able to transformations of the simulation domain geometry.

This work is organized as follows. Section 2 gives an overview on popular free open source mesh generation software packages and discusses their handling of mesh element size specification. Section 3 introduces ViennaMesh's element sizing framework, gives insight in its design and implementation, and discusses properties and features. The practicability of the

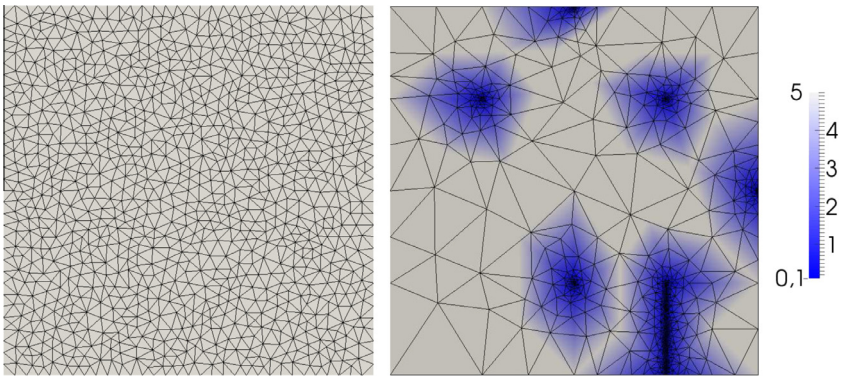


Fig. 1. Two different meshes of a square domain using different mesh element size specifications. The first mesh was generated using a constant global value for the mesh element size, while a scalar field was used for the second mesh. The mesh element size for the second mesh is visualized using the color. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

element sizing framework is shown by providing examples of meshes for simulations as well as a discussion of a geometry optimization application in Section 4.

2. Related work

In this section, popular mesh generation software packages are presented and their handling mesh element size specification is discussed. Due to their natural affinity with timely research work, we focus on a selection of free open source mesh generation software. An overview of these software packages and their mesh element size specification handling is given in Table 1.

The *Computational Geometry Algorithms Library (CGAL)* [10] is a free open source software library containing a rich collection of geometric algorithms written in C++, including a mesh generation algorithm for triangular and tetrahedral mesh generation. With the help of CGAL's oracle mechanism, meshes from arbitrary input geometries can be generated [13]. Because of its generic library design, CGAL supports a wide range of different mesh size specifications. Natively, a global value, a value for each mesh region, and a callback function are supported. Although it is not available within the CGAL package, an implementation for specifying the mesh element size as a scalar field is straightforward. In CGAL, the mesh element size itself is natively defined by the radius of the circumsphere of a triangle or tetrahedron. Due to its generic design, other definitions can easily be implemented.

Netgen [14] is a free open source mesh generation tool written in C++. It uses constructive solid geometries or triangular hull meshes to create tetrahedral meshes. Netgen supports a specification of the mesh element size by a global value and a value for each mesh region. Additionally, Netgen allows the user to provide a scalar field which defines the local mesh element size. The mesh element size is defined by the maximum dimension of a tetrahedron, which is equal to the size of its longest edge.

Tetgen [15] is a free open source tetrahedral mesh generation software written in C++ and is loosely based on the Triangle mesh generation tool. It creates tetrahedral meshes based on a so-called piecewise linear complex (PLC). A PLC is a geometry representation which consists of a set of polygons with holes, additional vertices, and lines. The mesh element size can be specified in Tetgen using a global value and one value per mesh region. Tetgen also supports callback functions for providing feedback whether certain tetrahedra are too large. However, the source code has to be recompiled to enable support for callback functions. The mesh element size is defined as the volume of a tetrahedron.

Triangle [8] is a free open source triangular mesh generation tool written in the C programming language. It generates triangular meshes based on a planar straight-line graph (PSLG), which is a collection of vertices, lines, and optional information about holes and mesh regions. Triangle supports a mesh element size specification via a global value and one value per mesh region. Additionally, callback functions can be provided to force refinement of certain triangles. The utilization of the

Table 1
An overview of the discussed meshing software and its handling of mesh element size specification.

	Global value	Global value for mesh regions	Scalar field	Callback function	Definition of the mesh element size
CGAL	•	•	•☆	•	Radius of the circumsphere of the tetrahedron
Netgen	•	•	•		Maximum linear dimension of the tetrahedron
Tetgen	•	•		•	Volume of the tetrahedron
Triangle	•	•		•	Volume of the triangle

☆ An implementation is required for specifying mesh element sizes using a scalar field.

callback function has the same issues as the callback function in the Tetgen package. The package use the area of the triangle to define the mesh element size.

3. ViennaMesh's element sizing framework

ViennaMesh [12,16] is a free open source software framework written in C++ providing infrastructure and unified interfaces for multiple different mesh related algorithms. ViennaMesh itself does not implement a mesh generation algorithm, but rather uses other free open source software packages, like Triangle or Tetgen. In this chapter the element sizing framework of the ViennaMesh software is presented. Section 3.1 introduces the design and the key concepts of the element sizing framework. Features of the element sizing framework and its technical implementation is discussed in Section 3.2, while Section 3.3 presents the configuration and usage of the element sizing framework.

3.1. Element sizing framework design

ViennaMesh's element sizing framework is based on three key concepts: An interchangeable module system allows selecting a mesh element size definition used for the mesh generation process. A tree of function objects which maps a specified location to the local mesh element size enables flexibility, interchangeability, and convenient usage. Geometric features of the simulation domain geometry can be used by these function objects to automatically adapt to the geometry to be meshed.

3.1.1. Mesh element size definition

As mentioned in Section 2, different mesh generation software packages use different definitions of the mesh element size. With a few exceptions, a conversion between different mesh element size definitions cannot be performed without assumptions about the shape of the element.

There are two different types of mesh element size definitions: volume-based and length-based. A volume-based definition, used for example in Triangle or Tetgen, has the property that the mesh element size scales with the reciprocal of the number of elements in the mesh. When the mesh element size using a volume-based definition is doubled, the number of mesh elements is approximately halved, independent on the geometric dimension of the simulation domain. Length-based definitions, like the radius of the circumsphere or the length of the longest edge, are used in software packages like CGAL or Netgen. Length-based definitions are independent of the dimension of the simulation domain, making the comparison between mesh elements of different dimension easier. Volume-based and length-based mesh size definitions only agree for one-dimensional geometries.

Depending on the scenario, either a volume-based or a length-based mesh element size definition is of advantage. Therefore, ViennaMesh's element sizing framework is not limited to one specific mesh element size definition but rather provides an interchangeable module system for several different mesh size definitions. These element size definition modules act as an interface between the element sizing framework and the mesh generation software by using the callback function mechanism of that software package and the function object tree to evaluate the local mesh element size. If element-based callback functions are supported, arbitrary mesh element size definitions can be implemented, otherwise only a module for the native mesh element size definition is provided. For software packages without support for callback functions but with support for scalar field mesh element size specification, a temporary scalar field is created and used for the mesh generation process. Only native mesh element size definitions are supported for these software packages.

3.1.2. Function object tree

The central entity of the element sizing framework is a tree of function objects where each function object can be evaluated at a given position in the simulation domain Ω to obtain an upper bound of the mesh element size at that specific location. There are two different types of function objects: leaf node function objects and internal node function objects.

Leaf node function objects calculate the initial element size at the requested location $x \in \Omega$, which are further processed by internal node function objects. A trivial representative of a leaf node function object is a constant function object which returns a configured value independent of the specified position. This is essentially the same as the global value mesh element size specification method. Another leaf node function object example, the local feature size, is a popular method for automatically calculating mesh element sizes. It is defined as the radius of the smallest circle with the center at $x \in \Omega$, which intersects any two disjoint vertices or lines of a PSLG [17]. Two other examples for leaf node function objects are the distance-to-interface function object and the scalar-field-gradient function object. The distance-to-interface function object calculates the shortest distance from the position to the interface of multiple mesh regions and can be used, for example, to obtain a fine mesh resolution near that interface. The scalar-field-gradient evaluates the gradient of a scalar field at the given position, which is useful, if a fine mesh resolution is desired in regions where the gradient of a scalar field is large, for example in regions with high changes in the doping profile of a semiconductor device [18]. Fig. 2 illustrates a selection of the presented leaf function objects.

Internal node function objects are used to modify or combine the mesh element size of other function objects to a resulting mesh element size function object. Internal node function objects can have an arbitrary number of child function object

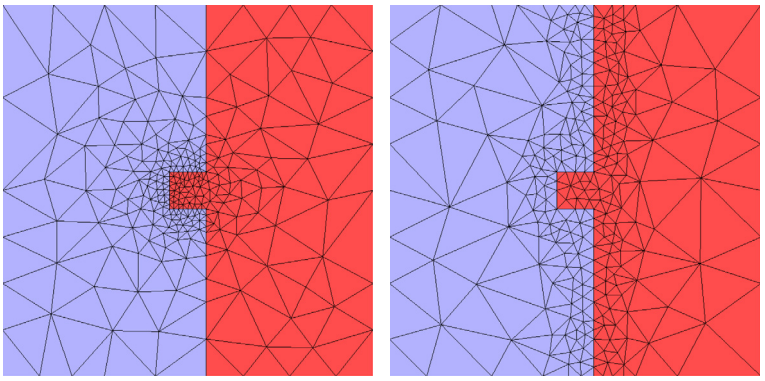


Fig. 2. Two meshes of a simple quadrilateral geometry with two mesh regions. The mesh regions are represented by the colors red and blue. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

nodes. One of the most important internal node function objects is the minimum function object, which calculates the minimum of other child node function objects. The minimum of multiple mesh object sizes at a specific location naturally combines them in a safe way. Often, different mesh element size specifications are necessary for different geometric regions within a simulation domain. ViennaMesh provides function objects which evaluate one specific child node function object based on the region of the requested position. Another example is the transformation of calculated mesh element sizes of child node function objects using an arithmetic expression defined by a scripting language like Python [19] or Lua [20]. An illustration of the presented internal node function objects is given in Fig. 3.

3.1.3. Geometric features

The element sizing framework provides an interface to query specific geometric features of the input domain geometry. These geometric features have to be declared in the configuration and can then be referenced in function objects. Geometric features are calculated only once and before the mesh generation process using a simple temporary mesh based on the input geometry. Depending on the definition of the mesh element size, either the volume or the radius of the circumsphere of the object can be used as a normalization value. For example, when using a volume-based mesh element size definition, a local mesh element size equal to the volume of the object divided by a desired number of mesh elements will result in a mesh with roughly that number of mesh elements. Another example for a geometric feature is the size of the interface between two mesh regions, which can be used to configure a lower bound for the mesh element count located at the interface. The two examples of geometric features are shown in Fig. 4.

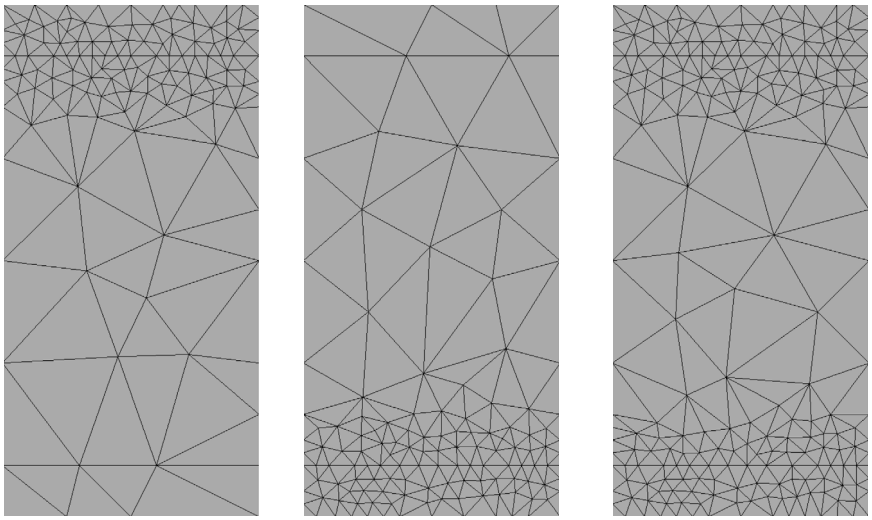
3.2. Implementation and features

As a part of ViennaMesh, the element sizing framework is written in C++ taking advantage of the Boost library [21], where Boost.Function and Boost.Bind are used to represent and manage the tree of functions objects.

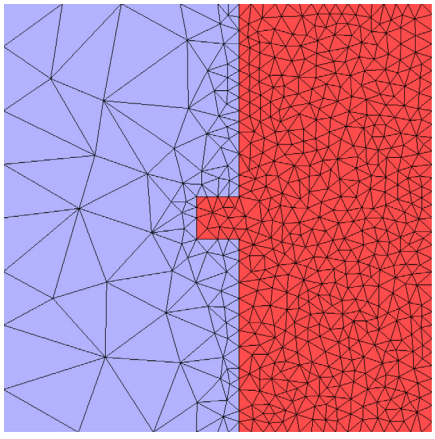
Geometric parameters and function objects, which operate on geometric information, like the distance-to-interface or region-based function objects, require a mesh of the input geometry. Before the actual mesh generation process is started, ViennaMesh uses the selected mesh generation module to automatically generate a temporary mesh based on the input geometry. This temporary mesh is utilized by the element sizing framework to calculate geometric parameters, like the volume of the mesh. It is also provided to function objects which need geometric information of the mesh, for example internal-node function objects which evaluate child function objects based on the region of the specific location. Therefore, the temporary mesh does not need to have either a good resolution, nor mesh elements of good quality. The mesh generation parameters are chosen in a way that the mesh generation is a fast as possible and the resulting mesh is as simple and coarse as possible while still being a good approximation of the geometry. There might be geometries, where ViennaMesh is not able to find parameters for the generation of the temporary mesh. In this case, a valid mesh for the geometry has to be provided manually.

Currently, the element sizing framework has interfaces to the following mesh generation modules in ViennaMesh: Triangle, Tetgen, and CGAL. Implementations for volume mesh element size definition as well as longest edge mesh element size definition are available for these three mesh generation modules. At this point, there is no implementation for the Netgen mesh generation module due to the lack of a callback function mechanism in the Netgen interface.

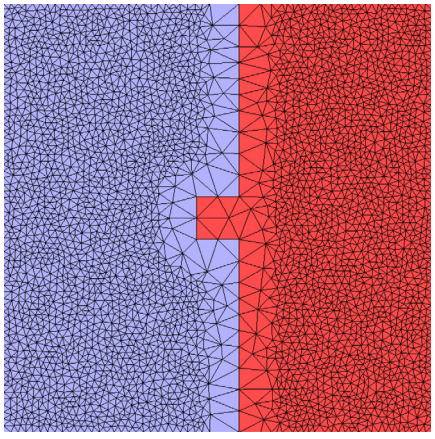
The concepts of the ViennaMesh element sizing framework enable mesh element size configurations which are invariant under a large set of transformations, especially deformations, of the simulation domain. To achieve this invariance, the set of possible transformations has to be kept in mind when creating the element size configuration. For example, if a minimum



Two meshes generated using a distance-to-interface function object to get small mesh element sizes at the top and the bottom, respectively and a mesh generated using the minimum function object to combine these two.



A mesh generated using different constant function objects for each mesh region.



A mesh generated using an arithmetic expression to invert the result of a distance-to-interface functor.

Fig. 3. Examples for the usage of the following internal node function objects: minimum function object, region-based function object, and arithmetic expression function object.

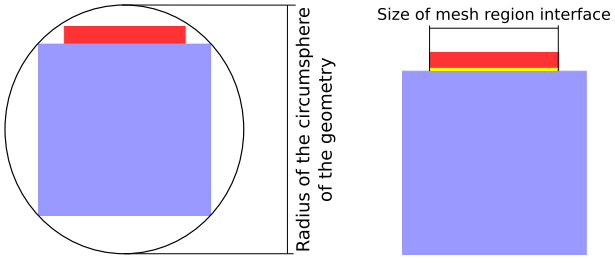


Fig. 4. Examples for geometric features.

number of elements on a region interface is required, the interface length of that interface has to be used rather than, for example, the radius of the circumsphere of the domain.

Depending on the properties of the metric used, distance based function objects provide invariance under certain transformations. For example, function objects based on the Euclidean distance are invariant under translation, rotation and mirroring operations. An invariance under scaling operations can be achieved using geometric features, like the volume or the circumradius of the mesh. The geometry independence can be seen in Fig. 5, where four meshes generated from similar geometries using the same mesh element size configuration are shown.

The invariance of the mesh element size configuration is especially of advantage when using parametric geometry definitions, like ViennaMesh's geometry templates [22]. Using the element sizing framework, only one single mesh element size configuration is needed for a specific geometry template instead of providing mesh element size configurations for each mesh generation separately.

3.3. Configuration and usage

ViennaMesh uses an eXtensible Markup Language (XML) scheme to configure the element sizing configuration, including its tree of function objects, the mesh element size definition used, as well as geometric features. The hierarchical structure of XML naturally reflects the structure of the tree of function objects making it a good choice for the configuration. Additionally, due to its human readability, it is convenient to use. Two main blocks have to be specified in an XML configuration: *geometric_features* and *function_object_tree*. In the *geometric_features* section, all geometric features used in function objects have to be declared and defined providing the name, the type, and properties for each geometric feature. The tree of function objects is defined in the *function_object_tree* section. For each function object its type and properties has to be provided. Geometric features can be referenced using the names specified in the *geometric_features* section.

The following listing presents a small example of the mesh element size configuration.

```
<size_configuration element_size_definition="longest_edge">
  <geometric_features>
    <feature name="MeshSize" type="circumradius">
      <object>mesh</object>
    </feature>
  </geometric_features>
  <function_object_tree>
    <function_object type="maximum">
      <function_object type="constant">
        MeshSize/100.0
      </function_object>
      <function_object type="distance_to_interface">
        <region>4</region>
        <region>2</region>
      </function_object>
    </function_object>
  </function_object_tree>
</size_configuration>
```

The mesh element size in this example is equal to the distance from the requested location to the interface of the mesh regions 4 and 2 but not smaller than a hundredth of the circumradius of the mesh. The longest edge mesh element size definition is chosen (Line 1) and one geometric feature with the name *MeshSize* being the circumradius of the mesh is declared and defined (Lines 2–6). The object of which the circumradius is calculated – in this case the mesh – is specified using the `<object>` tag (Line 4). The tree of function object is specified using a maximum function object (Line 9) as the root function object node. The child function objects are naturally embedded in the XML structure as child elements of the maximum function object. In this example, two child node function objects are used: a constant function object which references the previously declared geometric feature *MeshSize* (Line 10–12) and a distance-to-interface function object using the interface of the mesh region 4 and 2 (Line 13–16).

A mesh element size configuration is set up and can be passed to a ViennaMesh mesh generation module as an input parameter using ViennaMesh's application programming interface, as command line parameters, or via a Python module. The element sizing framework takes care of the XML parsing, the creation of the function object tree, and the interface to the mesh generation module using a callback mechanism. Fig. 6 illustrates a mesh generation workflow using a mesh element size configuration.

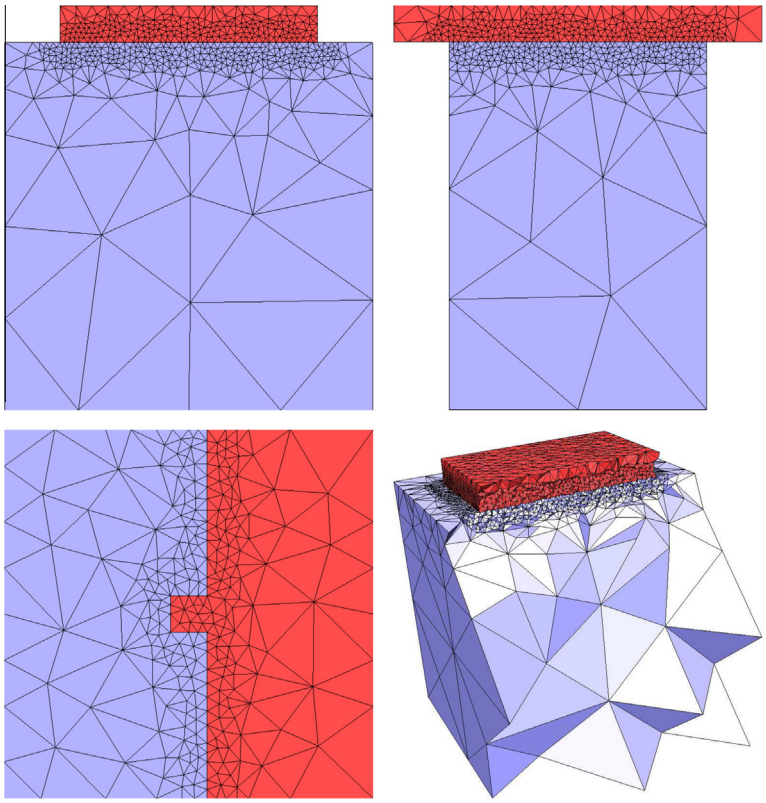


Fig. 5. Four different meshes with two mesh regions. All meshes are generated using the same mesh element size configuration based on a distance-to-interface function object. The fourth mesh shows the independence of the dimension. The longest edge mesh element size definition is used in these examples.

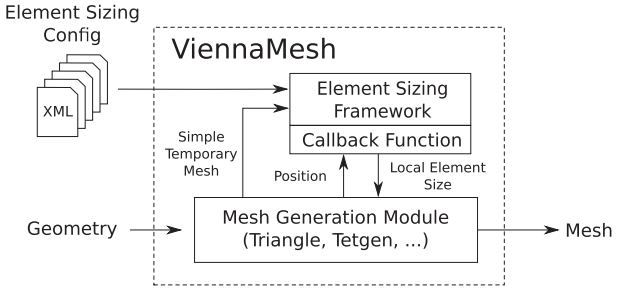


Fig. 6. The workflow of a mesh generation process. A mesh element size configuration is created and passed to the element sizing framework. The element sizing framework automatically assembles the function object tree and provides a callback function to the mesh generation module, which is used to query the local mesh element size based on a position. As described in Section 3.2, a simple mesh is generated by the mesh generation module and provided to the element sizing framework for, e.g., calculation of geometric features.

4. Examples and applications

In this chapter examples and applications using the element sizing framework are presented. A material-aware mesh generation process for a transistor device [23] is presented in Section 4.1. Section 4.2 covers a geometry optimization process taking advantage of both the element sizing framework as well as ViennaMesh's template-based geometry kernel.

4.1. Material-aware mesh generation of silicon-based devices

In the field of semiconductor device simulation, the characteristics of a device highly depends on its doping profile which is often generated by numerical semiconductor process simulations [1]. The doping profile of a device, which can be

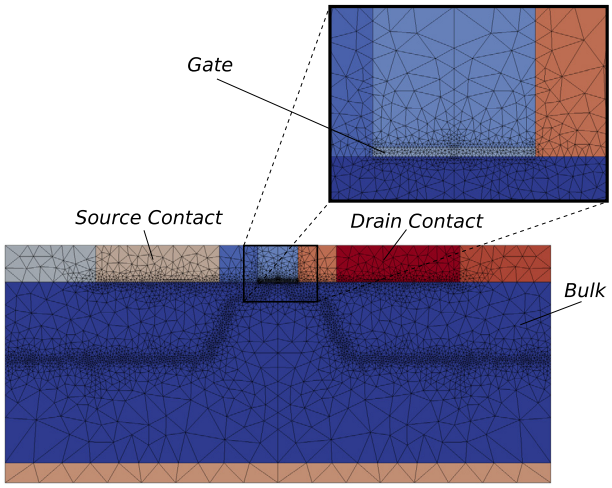


Fig. 7. A mesh of an n-channel laterally diffused metal oxide semiconductor transistor device. A high mesh resolution is achieved in and near the *Gate* area and on the interfaces between the *Bulk* region the the two contacts. Within the *Bulk* region, the doping profile is used to obtain a fine mesh resolution in areas of high doping profile changes.

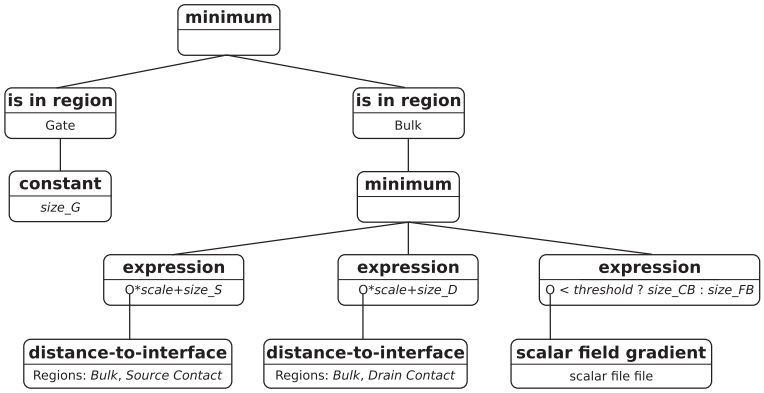


Fig. 8. The function object tree for the n-channel laterally diffused metal oxide semiconductor transistor device in Fig. 7. On the top, a minimum function object combines the function object sub-trees for the *Gate* region and the *Bulk* region. In the *Gate* region, a constant mesh element size is used. In the *Bulk* region, another minimum function object combines distances to the contact interfaces and the scalar field gradient, which are modified using expression function objects. The sizes in the constant and the expression function objects are pre-defined using geometric parameters. *size_G* is equal to the minimal mesh element size. *size_S* and *size_D* are defined as the interface length of the *Source* and *Drain* contact divided by the desired number of elements on that interface. *size_CB* and *size_FB* are used if the gradient is lower or higher than *threshold*, respectively. *scale* defines the grading from small to large mesh elements.

represented by scalar fields, has significant impact on the convergence behavior of numerical simulations such as the deterministic solution of the Boltzmann transport equation [23]. Especially in regions of high doping profile changes, a fine mesh resolution is of advantage for the simulation process. Additionally, the convergence behavior of the numerical simulation benefits from a high mesh resolution in and near the gate area and on the contact interfaces.

Fig. 7 shows an n-channel laterally diffused metal oxide semiconductor transistor device meshed with a mesh element size configuration fulfilling these requirements. The function object tree of this mesh element size configuration is visualized in Fig. 8. On the top, a minimum function object combines the local sizes of the *Gate* and the *Bulk* region. In the *Gate* region, a constant mesh element size is used. In the *Bulk* region, another minimum function object is used to combine the two distances to the *Source* and *Drain* contact interface and the scalar field gradient. The two distances and the scalar field gradient output are modified using expression function objects. The resulting mesh has 7454 elements and 4153 vertices. In contrast, a mesh with a constant element size in the *Bulk* and *Gate* region equal *size_G* has 281,152 elements and 142,284 vertices. The triangle mesh generation module and a length-based mesh element size definition have been used to generate these meshes.

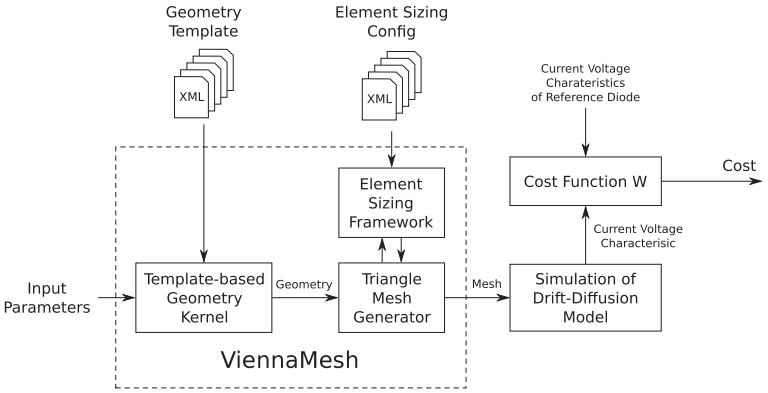


Fig. 9. The function $F : \mathbb{R}^n \to \mathbb{R}$ and its components. At first, a geometry is created using the ViennaMesh template-based geometry kernel and the input parameters. A mesh is generated based on the resulting geometry using a mesh generation module. The mesh element size configuration is static and does not act as an input to the function F due to its independence of the geometry. A simulation is performed using the generated mesh and finally a cost function W calculates the Euclidean distance between the current values of the simulated current–voltage characteristics and the current–voltage characteristics of a reference diode.

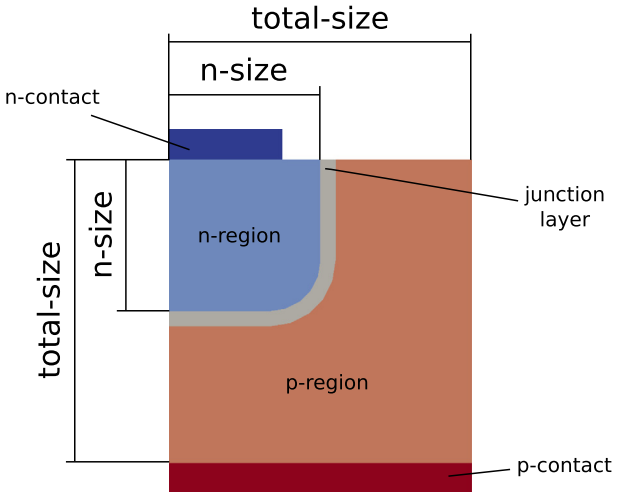


Fig. 10. The geometry template of a two-dimensional diode device. The n- and p-region indicate regions with high n or p doing, respectively. A junction layer has been inserted between these two regions to improve simulation performance. The template only provides one input parameter, being the size of the n-region *n-size*.

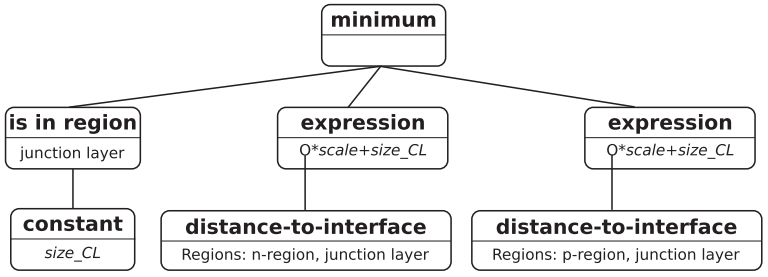


Fig. 11. The function object tree for the pn-junction diode in Fig. 10. On the top, a minimum function object combines the function object sub-trees for the junction layer, the n- and p-region. In the junction layer, a constant mesh element size is used. For the other regions, the distance to the interfaces of the junction layer and the n- and p-region is modified using expression objects which naturally result in a smooth grading from small to large mesh elements. *size_CL* is the desired minimum mesh element size in the junction layer and *scale* is used to define the grading.

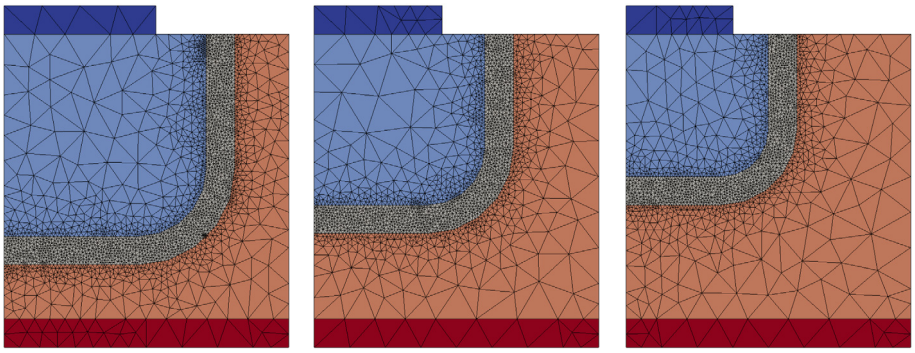


Fig. 12. Three examples meshes generated in the optimization process. The same mesh element size configuration has been used for all three meshes.

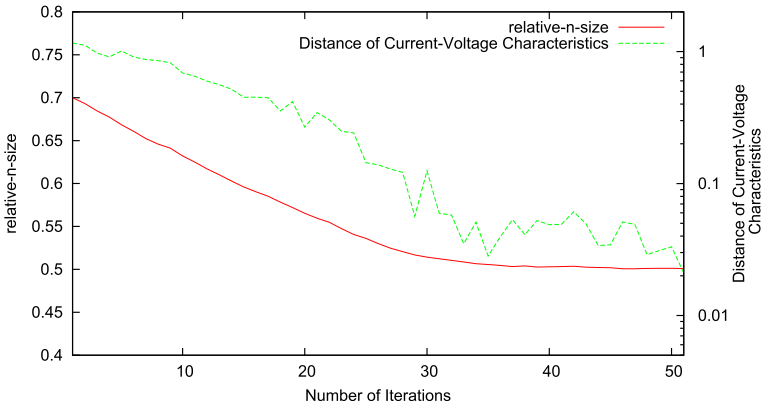


Fig. 13. The convergence behavior of the optimization process. The solid red graph illustrates the size of the n-region *n-size* while the dashed green graph shows the distance of the current–voltage characteristics for each iteration. It can be seen, that the geometry converges to the reference diode device geometry with *relative-n-size* = $\frac{1}{2}$. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this article.)

4.2. Geometry optimization

In this example we use ViennaMesh's element sizing framework to optimize a geometry which is based on a parametric definition. An optimization process is performed to find the parameters for a geometry template of a pn-junction diode so that the current–voltage characteristic of that junction matches the characteristic of a reference diode. Therefore, we define a function F which maps the input parameters of the geometry template to the l^2-distance between the current–voltage characteristics of the reference diode and the diode based on the geometry template. Classical optimization methods, like the Newton method [24], can be used to minimize the function F.

The function F is defined as the concatenation of the template-based geometry kernel, a mesh generation module, a simulation, and a cost function W. ViennaMesh's template-based geometry kernel uses the input parameters and a geometry template to create a geometry which is used by the mesh generation module to generate a mesh. The element sizing framework is utilized for specifying a mesh element size configuration, which is invariant under deformations based on the geometric parameters of geometry template. Due to the independence of the geometry, only one mesh element size configuration has to be provided for a specific input geometry template. We further use a finite volume method for solving the boundary value problem governed by the equations of a drift–diffusion model [1]. Finally, the cost function W is defined as the Euclidean distance between the current values of the current–voltage characteristics of the solution and the reference current–voltage characteristics. The function F is illustrated in Fig. 9.

The geometric domain of the diode device has been modeled using a two-dimensional geometry template illustrated in Fig. 10. Due to a high gradient in the doping profile in and near the junction layer, a fine mesh resolution is beneficial in this area. The mesh element size specification used in this application is visualized in Fig. 11. Some example meshes generated in the optimization process are shown in Fig. 12 For simplicity's sake, only one geometric parameter – the relative size of the n-region *relative-n-size* = *n-size*/*total-size* – is provided. Based on the function F, a gradient descent method [24] is used to find the optimal value for *relative-n-size*. The gradient, required in the optimization process, is evaluated using numerical differentiation. Fig. 13 shows the convergence behavior of the optimization process using a reference diode with *relative-n-size* = $\frac{1}{2}$.

As expected, the input parameter *relative-n-size* of the optimization process converges to $\frac{1}{2}$, implying a convergence of the geometry to the geometry of the reference diode.

5. Conclusion

The specification of local mesh element sizes is important for the convergence behavior of simulation processes. ViennaMesh's element sizing framework, a convenient and flexible interface to specify local mesh element sizes, has been presented. It has been shown that the design supports local mesh element size specifications which are independent of the geometry enabling re-usage for similar object geometries. Possibilities and advantages have been discussed, especially in combination with ViennaMesh's template-based meshing kernel. We provided two examples in the field of semiconductor device simulation and a geometry optimization application to show the practicability as well as the flexibility of the element sizing framework.

Acknowledgments

This work has been supported by the European Research Council (ERC), Grant #247056 MOSILSPIN and by the Austrian Science Fund FWF, Grants P23296 and P23598.

References

[1] S. Selberherr, Analysis and Simulation of Semiconductor Devices, Springer-Verlag, Wien – New York, 1984.
[2] D.J. Acheson, Elementary Fluid Dynamics, Clarendon Press, 1990.
[3] H. Edelsbrunner, Geometry and Topology for Mesh Generation, Cambridge University Press, 2001.
[4] S.-W. Cheng, T.K. Dey, J.R. Shewchuk, Delaunay Mesh Generation, CRC Press, 2013.
[5] J.R. Shewchuk, What is a good linear finite element? – Interpolation, conditioning, anisotropy, and quality measures, in: Proceedings of the 11th International Meshing Roundtable, 2002, pp. 115–126.
[6] P. Solin, K. Segeth, I. Dolezel, Higher-order finite element methods, CRC Press, 2003.
[7] S. Barraud, M. Berthome, R. Coquand, M. Casse, T. Ernst, M.P. Samson, P. Perreau, K. Bourdelle, O. Faynot, T. Poiroux, Scaling of trigate junctionless nanowire MOSFET with gate length down to 13 nm, Electron Device Lett., IEEE 33 (9) (2012) 1225–1227, http://dx.doi.org/10.1109/LED.2012.2203091.
[8] Triangle – A two-dimensional quality mesh generator and Delaunay triangulator (Sep. 2013). <http://triangle.org/>.
[9] Tetgen – A quality tetrahedral mesh generator and a 3d Delaunay triangulator (Sep. 2013). <http://tetgen.org/>.
[10] CGAL – Computational geometry algorithms library (Sep. 2013). <http://www.cgal.org/>.
[11] Netgen mesh generator (Sep. 2013). <http://sourceforge.net/projects/netgen-mesher/>.
[12] ViennaMesh (Sep. 2013). <http://viennamesh.sourceforge.net/>.
[13] C. Jamin, P. Alliez, M. Yvinec, J.-D. Boissonnat, CGALmesh: A generic framework for delaunay mesh generation, Research Report 8256, INRIA (2013).
[14] J. Schöberl, NETGEN – An advancing front 2D/3D-mesh generator based on abstract rules, Comput. Visual. Sci. 1 (1997) 41–52, http://dx.doi.org/10.1007/s007910050004.
[15] H. Si, Three dimensional boundary conforming Delaunay mesh generation (Dissertation), Technische Universität Berlin (2008).
[16] F. Rudolf, J. Weinbub, K. Rupp, S. Selberherr, The meshing framework ViennaMesh for finite element applications, J. Comput. Appl. Math. 270 (0) (2014) 166–177, http://dx.doi.org/10.1016/j.cam.2014.02.005 (Fourth International Conference on Finite Element Methods in Engineering and Sciences (FEMTEC 2013)).
[17] J. Ruppert, A Delaunay refinement algorithm for quality 2-dimensional mesh generation, J. Algorithms 18 (3) (1995) 548–585, http://dx.doi.org/10.1006/jagm.1995.1021.
[18] P. Fleischmann, S. Selberherr, Enhanced advancing front Delaunay meshing in TCAD, in: 2002 International Conference on Simulation of Semiconductor Processes and Devices, 2002, pp. 99–102, talk: International Conference on Simulation of Semiconductor Processes and Devices (SISPAD), Kobe, 2002-09-04–2002-09-06.
[19] Python (Sep. 2013). <https://www.python.org/>.
[20] The programming language Lua (Sep. 2013). <www.lua.org/>.
[21] Boost (Sep. 2013). <http://www.boost.org/>.
[22] F. Rudolf, J. Weinbub, K. Rupp, A. Morhammer, S. Selberherr, Template-based mesh generation for semiconductor devices, in: Proceedings of the 2014 International Conference on Simulation of Semiconductor Processes and Devices, 2014, pp. 217–220.
[23] Y. Wimmer, S. Tyaginov, F. Rudolf, K. Rupp, M. Bina, H. Enichlmair, J.-M. Park, R. Minixhofer, T. Grasser, Physical modeling of hot-carrier degradation in nLDMOS transistors, in: Proc. International Integrated Reliability Workshop (IIRW), 2014.
[24] J. Snyman, Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms, Springer, 2005.