Template-Based Mesh Generation for Semiconductor Devices

Florian Rudolf*, Josef Weinbub*, Karl Rupp*†, Andreas Morhammer*‡, and Siegfried Selberherr*
* Institute for Microelectronics, TU Wien, Vienna, Austria
† Institute for Analysis and Scientific Computing, TU Wien, Vienna, Austria
‡Christian Doppler Laboratory for Reliability Issues in Microelectronics at the Institute for Microelectronics
{rudolf|weinbub|rupp|morhammer|selberherr}@iue.tuwien.ac.at

***Abstract*—Creating multiple meshes of a semiconductor device by varying specific geometric properties, like the gate length of a MOSFET, is a crucial step for optimization or scaling processes of these devices. A geometry generation technique for semiconductor devices using geometry templates is presented and implemented in the open source meshing tool ViennaMesh, providing a convenient mechanism for creating device geometries based on a selected set of parameters. These geometries can be used by ViennaMesh to create high-quality meshes to be exported and used by simulation tools. Results of meshes for two-dimensional MOSFET and three-dimensional FinFET devices created by this technique are presented.**

I. Introduction

Device optimization and device scaling is an important topic for technology computer aided design (TCAD) of semiconductor devices [1], [2], for which techniques like the finite element method or the finite volume method are used to simulate physical properties in order to predict a device's behavior. Most of these simulation processes require a discretization of the *geometry* of the device, which defines its shape and size. The generation process of such a discretization, called *mesh*, is a crucial step for these simulations [3], [4]. This work focuses on the automatic creation of device geometries required by mesh generation algorithms, but also covers the mesh generation workflow required for device simulation.

In device optimization and device scaling processes often just a few geometric parameters are of interest. For example, the geometric parameters gate length, gate width, and oxide thickness are of importance for MOSFET devices. Nevertheless, changing a specific geometric feature generally affects other parts of the geometry. For example, changing the edge length of a simple cube will affect the location of all 8 vertices. Thus, with classical geometry representations, like boundary representations, all parts of the geometry influenced by the geometric feature changes have to be recreated manually according to these changes.

The concept of geometry templates has been applied to closed source software, like COMSOL Multiphysics [5] or Synopsys Sentaurus Structure Editor [6], but often limited to the meshing algorithm provided, which decreases the flexibility of the mesh generation process. Additionally, the geometry templates usually support just one single scripting language, thus only impeding re-use of already implemented geometric algorithms. Also, a couple of open source meshing tools are available, like Triangle [7], the Computational Geometry Algorithms Library [8], [9], Tetgen [10], or Netgen [11]. The supported types of input geometries for these meshing tools, typically boundary representation geometries or constructive solid geometries (CSGs), do not natively provide a workflow for changing specific geometric features, like the gate length of a MOSFET device, decreasing convenience and flexibility, if certain features have to be changed repeatedly. Another issue with available meshing tools is the incompatibility of their geometry data formats. When using multiple meshing tools, e.g., for mesh element quality optimization, the device geometry has to be created for each tool and for each parameter set separately.

II. Template-Based Geometry Kernel

We tackle these issues by providing a template-based geometry kernel for our meshing framework ViennaMesh [12], which is able to generate geometries based on a geometry template and a set of input parameters. Similar to a CSG, a geometry template is defined by a set of simple geometric entities, like rectangles or cubes, and a hierarchical tree of boolean operations for combinations of these entities. In contrast to classical CSG, the parameters of a geometric entity, like the center of a cube, do not have to be specified explicitly, but are also allowed to depend on other values. For this purpose, support for scalar or vector input parameters as well as for temporaries has been added, on which the parameters of the geometric entities can depend. These dependencies as well as the temporaries can be defined by arithmetic expressions using the input parameters or other temporaries.

A schematic of a geometry template of a simple device geometry, a pn-diode, is shown in Figure 1. A more practical example is given in Figure 2, where a two-dimensional geometry template of a MOSFET device providing the gate length as well as the oxide thickness as input parameters is presented. Additionally, restrictions to input parameters can be specified, which enable validation of the input parameter values. The eXtensible Markup Language (XML) is used to define a template, its geometric entities, boolean operations, temporaries, and input parameters together with their optional default values and optional restrictions.

The template-based geometry kernel includes a mechanism which provides an interface to evaluate arithmetic expressions. To increase the flexibility, the abstract design of the expression evaluation mechanism enables multiple evaluation backends for different expression or scripting languages, like the interpreters for the Python language [13] or the Lua language [14]. The expression evaluation mechanism is used by the template-based geometry kernel to evaluate the arithmetic expressions of the temporaries and the parameters of geometric entities to create a resulting geometry in CSG representation.

978-1-4799-5288-5/14/$31.00

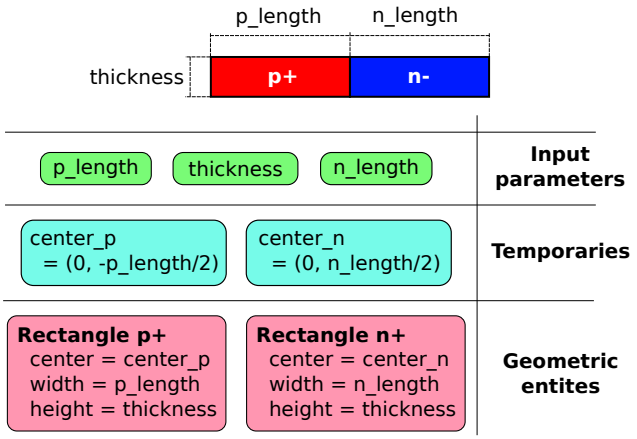


Fig. 1: A scheme of a two-dimensional geometry template of a pn-diode. Three input parameters are provided: the length of the n segment *length_n*, the length of the p segment *length_p*, and the *thickness*. Two temporaries, the centers of the mesh segment rectangles, are calculated based on these input parameters. Each mesh segment is specified with a rectangle using the temporaries as centers and the lengths and thickness as width and height, respectively.

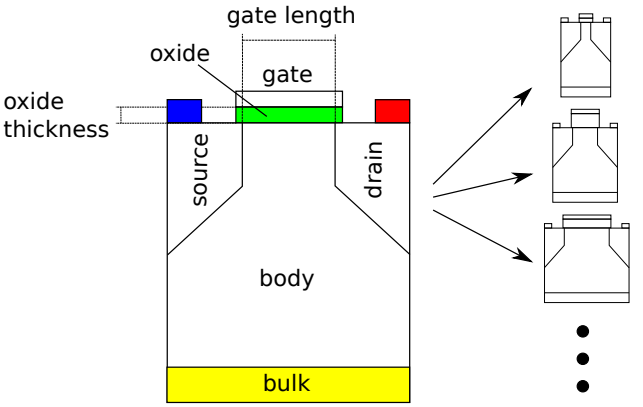


Fig. 2: A two-dimensional MOSFET device template, providing the features gate length and the oxide thickness, as well as some geometries for different input parameters.

One can distinguish between two types of TCAD software tool users: *end users*, who utilize software tools and interprets the software as a *black box*, and developers [15]. In our approach it is possible to provide these ready-to-use device templates to end users, whereas the templates are implemented by developers. Accessing the input parameters is supported using ViennaMesh's application programming interface (API), as command line parameters, or via its Python module. Additionally, the API can be used by a graphical user interface to access the input parameters. Figure 3 shows the workflow of the template-based geometry kernel with its input parameters and the generated output geometry.

III. Template Creation and Meshing Workflow

The template-based geometry kernel is well integrated in the ViennaMesh meshing framework and is therefore easily combinable with other ViennaMesh algorithms. Particularly,

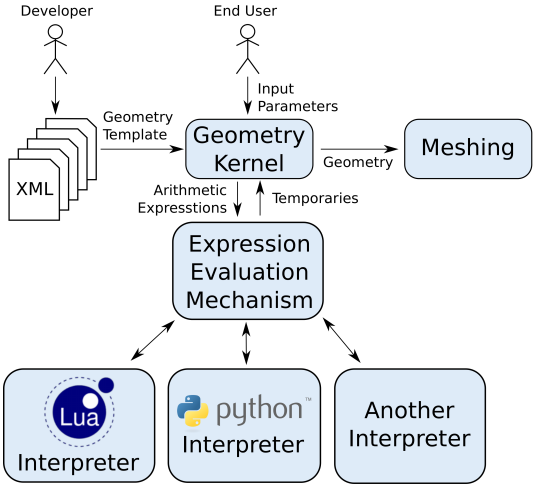


Fig. 3: The workflow of the template-based geometry kernel with its input parameters and the output geometry is shown. The expression evaluation mechanism utilizes an interpreter backend to evaluate the expressions defined in the input geometry template using a set of input parameters by the end user. The geometry template itself is created by a developer, while the end user sets the input parameters to create a meshable geometry.

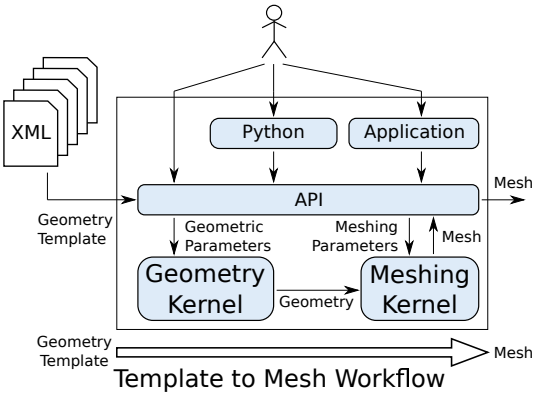
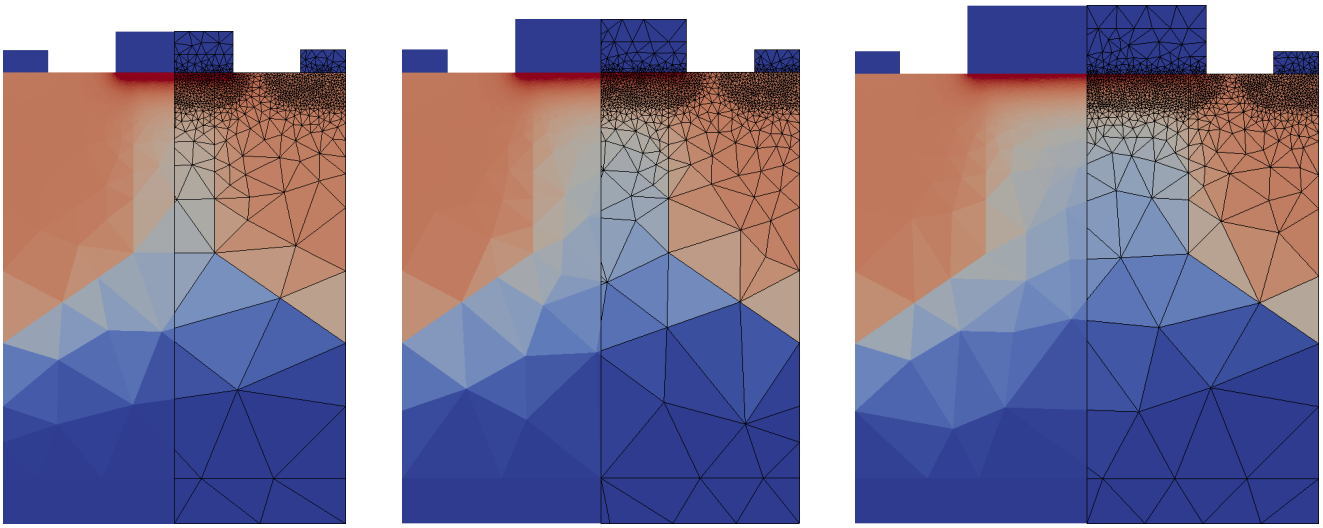
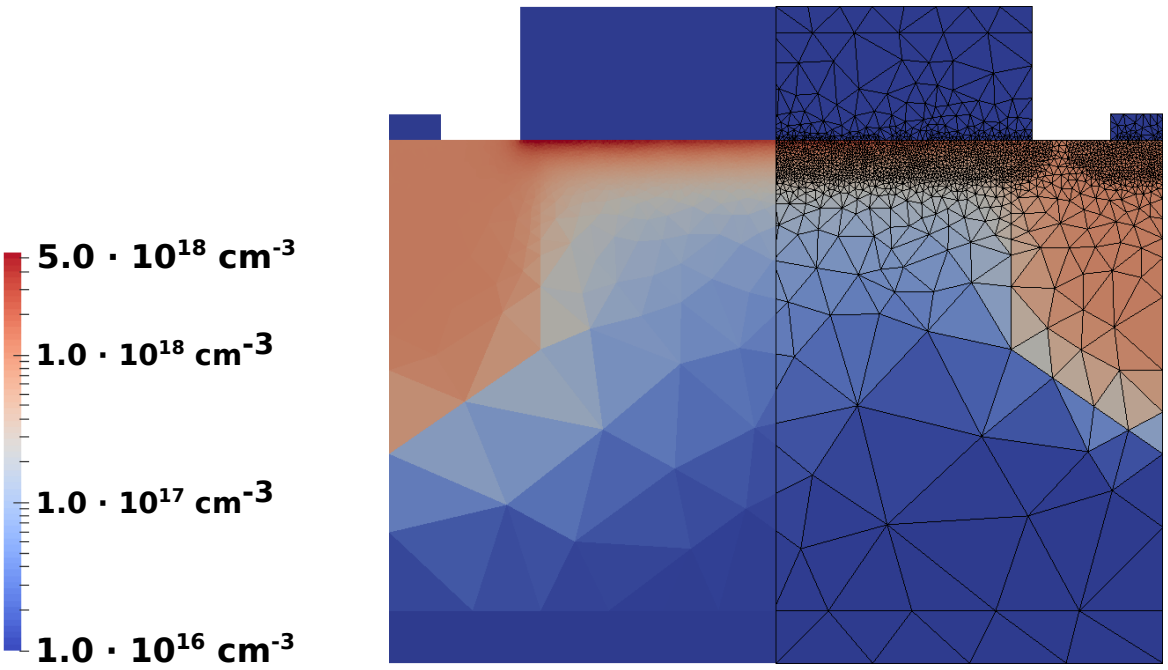


Fig. 4: The internal workflow of ViennaMesh's template-based meshing process is presented. A geometry template together with parameter values is used to create a geometry which is further processed by a meshing kernel to generate a mesh.

geometries created by this kernel can further be used by the ViennaMesh meshing algorithms to generate a mesh. To ensure good simulation results in critical device regions, such as the channel of a MOSFET, a mechanism to control the local element sizes is necessary. ViennaMesh's element sizing framework [16] provides geometry-independent control of local mesh element sizes, which is especially important when working with similar but slightly different geometries. Figure 4 gives an overview of the workflow with the template-based geometry kernel. Using this workflow, an end user is able to conveniently obtain meshes by loading a geometry template, optionally setting desired input parameters, and triggering the meshing process via, e.g., the ViennaMesh Python module. Exporting the resulting mesh to, e.g., a VTK file, finally permits its use by other simulation or visualization tools.



MOSFET, 18nm gate length MOSFET, 30nm gate length MOSFET, 45nm gate length



MOSFET, 90nm gate length

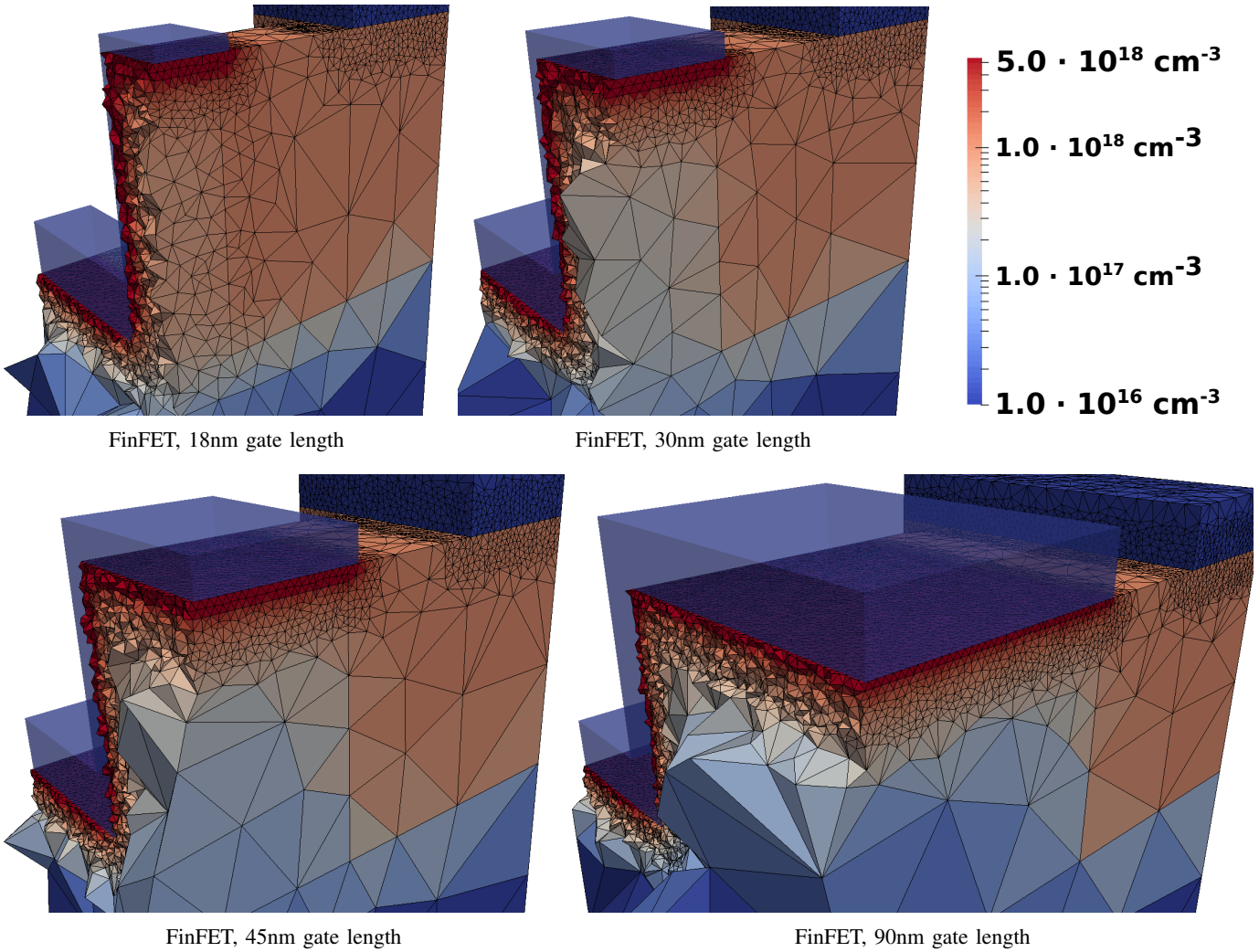
Fig. 5: Electron concentration in two-dimensional MOSFET devices with 18nm, 30nm, 45nm, and 90nm gate length.

IV. Examples

ViennaMesh's template mechanism has been applied to a constant-field scaling process for two-dimensional MOSFET and three-dimensional FinFET devices. Geometry templates for the required devices have been created, providing the gate length and the oxide thickness as input parameters. The three-dimensional FinFET device template additionally provides the gate width as an input parameter. These templates were used by the template-based geometry kernel to create geometries for different input parameter sets. The geometries were meshed utilizing ViennaMesh's element sizing framework to achieve desired local mesh element sizes, especially important for the channel regions. This is illustrated in simple drift-diffusion simulations using the finite volume method as presented in Figure 5 and Figure 6 for two-dimensional MOSFET devices and for three-dimensional FinFet devices, respectively. Devices with gate lengths of 18nm, 30nm, 45nm, and 90nm were simulated.

V. Summary

We presented the template-based geometry kernel of the open source library ViennaMesh, its mechanism to create meshes from an XML description of a parameterized family of similar geometries and its convenient scripting interfaces, e.g., for the Python language.



FinFET, 18nm gate length

FinFET, 30nm gate length

FinFET, 45nm gate length

FinFET, 90nm gate length

Fig. 6: Electron concentration in sliced three-dimensional FinFET devices with 18nm, 30nm, 45nm, and 90nm gate length.

This approach simplifies automatic processes where specific geometric features of the device geometry are changed repeatedly, for example, optimization or scaling processes. Results of a drift-diffusion simulation of two-dimensional MOSFET and three-dimensional FinFET devices with different gate lengths were presented.

VI. Acknowledgements

This work has been supported by the European Research Council, grant #247056 MOSILSPIN and by the Austrian Science Fund FWF, grants P23296 and P23598.

References

[1] International Technology Roadmap for Semiconductors, http://public.itrs.net/

[2] S. Barraud et al., Scaling of Trigate Junctionless Nanowire MOSFET With Gate Length Down to 13 nm, IEEE Electron Device Letters, 2012

[3] S. Selberherr, Analysis and Simulation of Semiconductor Devices, Springer-Verlag, Wien - New York, 1984.

[4] P. Fleischmann et al., Mesh Generation for Application in Technology CAD, IEICE Transactions on Electronics, 1999

[5] COMSOL Multiphysics, http://www.comsol.com/

[6] Synopsys Sentaurus Structure Editor, http://www.synopsys.com/Tools/TCAD/Pages/StructureEditor.aspx

[7] J. Shewchuk, Triangle: Engineering A 2D Quality Mesh Generator and Delaunay Triangulator, Applied Computational Geometry Towards Geometric Engineering, 1996

[8] Computational Geometry Algorithms Library, http://www.cgal.org/

[9] C. Jamin et al., CGALmesh: a Generic Framework for Delaunay Mesh Generation, INRIA Research Report 8256, 2013

[10] Tetgen, http://tetgen.org/

[11] J. Schöberl, NETGEN - An Advancing Front 2D/3D-Mesh Generator Based On Abstract Rules, Computing and Visualization in Science, 1997

[12] F. Rudolf et al., The Meshing Framework ViennaMesh for Finite Element Applications, Journal of Computational and Applied Mathematics, 2014

[13] Python, https://www.python.org/

[14] Lua, https://www.lua.org/

[15] J. Weinbub, Frameworks for Micro- and Nanoelectronics Device Simulation, Dissertation TU Wien, 2014

[16] F. Rudolf et al., Mesh Generation Using Dynamic Sizing Functions, European Seminar on Computing, 2014