Free Open Source Mesh Healing for TCAD Device Simulations

Florian Rudolf[1](✉), Josef Weinbub[1], Karl Rupp[1,2], Peter Resutik[1], Andreas Morhammer[3], and Siegfried Selberherr[1]

[1] Institute for Microelectronics, TU Wien, Vienna, Austria
{rudolf,weinbub,resutik,selberherr}@iue.tuwien.ac.at
[2] Institute for Analysis and Scientific Computing, TU Wien, Vienna, Austria
rupp@iue.tuwien.ac.at
[3] Christian Doppler Laboratory for Reliability Issues in Microelectronics, Institute for Microelectronics, TU Wien, Vienna, Austria
morhammer@iue.tuwien.ac.at

Abstract. Device geometries in technology computer-aided design processes are often generated using parametric solid modeling computer-aided design tools. However, geometries generated with these tools often lack geometric properties, like being intersection-free, which are required for volumetric mesh generation as well as discretization methods. Contributing to this problem is the fact, that device geometries often have multiple regions, used for, e.g., assigning different material parameters. Therefore, a *healing* process of the geometry is required, which detects the errors and repairs them. In this paper, we identify errors in multi-region device geometries created using computer-aided design tools. A robust algorithm pipeline for *healing* these errors is presented, which has been implemented in ViennaMesh. This algorithm pipeline is applied on complex device geometries. We show, that our approach robustly *heals* device geometries created with computer-aided design tools and is even able to handle certain modeling inaccuracies.

1 Introduction

Many commercial parametric solid modeling computer-aided design (CAD) tools, like AutoCAD [1], are available and also various free open source tools, like FreeCAD [2], are used in many applications. Some technology computer-aided design (TCAD) simulation suites have modules for CAD processing, but these modules are usually not as powerful as their standalone counterparts. For example, Synopsys Sentaurus TCAD provides a structure editor for modeling device geometries [3]. In contrast, many free open source TCAD tools, like DEVSIM [4], lack the CAD processing module and, therefore, they require a ready-to-simulate input mesh representing the device geometry.

Regardless of the utilized CAD tools, the task of generating a ready-to-simulate mesh based on a CAD-based geometry is challenging. The finite difference method is particularly attractive whenever the domain can be represented

I. Lirkov et al. (Eds.): LSSC 2015, LNCS 9374, pp. 293–300, 2015.
DOI: 10.1007/978-3-319-26520-9_32

well with a structured grid, possibly taking additional smooth geometric transformations into account to allow for more complicated domains [5]. In other cases, the finite element method or the finite volume method are popular choices. However, these methods require the mesh to be conforming and valid [6]. Additionally, simulations often require the mesh to be partitioned into several regions, which can, for example, be used to locally assign material properties.

To generate a volumetric mesh which can be used in a simulation, the device geometry created with the CAD tool has to be exported. Usually, a widely supported geometry representation is chosen for this export in order to have a high degree of freedom when selecting the volumetric mesh generation tool. Popular geometry representation formats, like the standard for the exchange of product model data (STEP, ISO 10303) [7], provide a rich feature set, but they are not supported by a variety of popular open source mesh generation tools, like Tetgen [8]. On the other hand, triangular hull geometry representations, like StereoLithography (STL) [9], do not provide a high level of flexibility, but are supported by a large number of mesh generation tools. Triangular hull geometry representations, however, might have topological issues like duplicate elements, or geometrical errors like self-intersections, gaps, or holes. These errors have to be *healed* before a mesh generation can be performed. Additionally, STL and similar geometry representations lack support for multiple mesh regions.

In this work we present an algorithm pipeline which robustly *heals* errors in multi-region triangular hull geometries of complex device structures exported by CAD tools. Section 2 discusses possible errors in triangular hull geometry representations and provides an overview of research in mesh *healing*. The algorithm pipeline for *healing* the triangular hull geometry and generating a mesh is presented in Sect. 3. This algorithm pipeline is implemented in the free open source meshing tool ViennaMesh [10]. The pipeline is applied to example devices created with FreeCAD in Sects. 4 and 5 summarizes the work and gives an outlook for future work.

2 Background and Related Work

When CAD tools export geometries as triangular hulls, a discretization of the geometry has to be performed for the non-planar surfaces. Due to inaccuracies during the modeling process or different discretizations of interfaces, the resulting triangular hull might have topological or geometrical errors. Relevant triangular hull errors are listed below and visualized in Fig. 1 [11,12].

- **Duplicate vertices and elements** are vertices or elements which occur more than once in the mesh.
- **Isolated and dangling elements** are vertices and lines which are not edges or vertices of any hull triangle.
- **Singular edges** occur, when an edge is shared by more than two triangles.
- **Singular vertices** occur, when a vertex is shared by two unrelated sets of triangles.

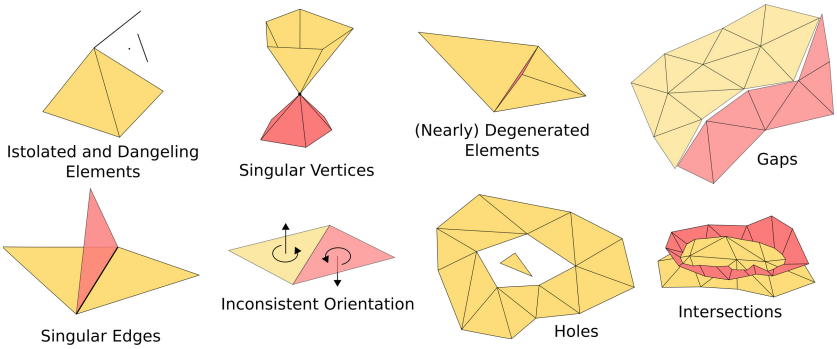

Fig. 1. Visualization of all triangular hull mesh errors except duplicate vertices and elements.

- **Inconsistent orientation** occurs, when two neighboring triangles have different vertex orientations.
- **Nearly degenerated elements** are triangles which are degenerated or nearly degenerated, meaning their surface area is very small compared to their edge lengths.
- **Holes** occur, when a hull is not fully closed.
- **Gaps** occur, when two different hulls are not topologically connected to each other.
- **Intersections** occur, when a triangle intersects another triangle.

Duplicate vertices can be *healed* by merging them and duplicate elements can safely be removed. Isolated and dangling elements can be identified and removed using topological operations. In many file formats, like STL, isolated and dangling vertices and lines cannot occur because vertices and lines are not stored explicitly. Due to the fact that many mesh *healing* algorithms originate in the field of computer graphics, the definition of a valid *healed* mesh is different to the definition of a *healed* mesh for volumetric ready-to-simulate mesh generation. In particular, a singular edge might be valid for multi-region geometries, because it can be an interface edge between two different mesh regions. Similarly, a singular vertex might also be valid, but can lead to numerical issues during the simulation. Therefore, singular vertices must be detected using topological operations and split up into multiple new vertices, one for each triangle set. If the volumetric mesh generation algorithm requires consistent orientations, like most advancing front mesh generation algorithms [13] do, they can be fixed by vertex index swapping of triangles with wrong orientation. Degenerated or nearly degenerated triangles can be fixed by either performing an edge collapse [14], if two vertices are close to each other, or re-meshing the area of the degenerated triangle and its three neighbors.

The other three types of errors, being holes, gaps, and intersections, are much more challenging to *heal*. Many different algorithms have been developed, which address these types of errors [11,12,15]. Several open and closed source mesh

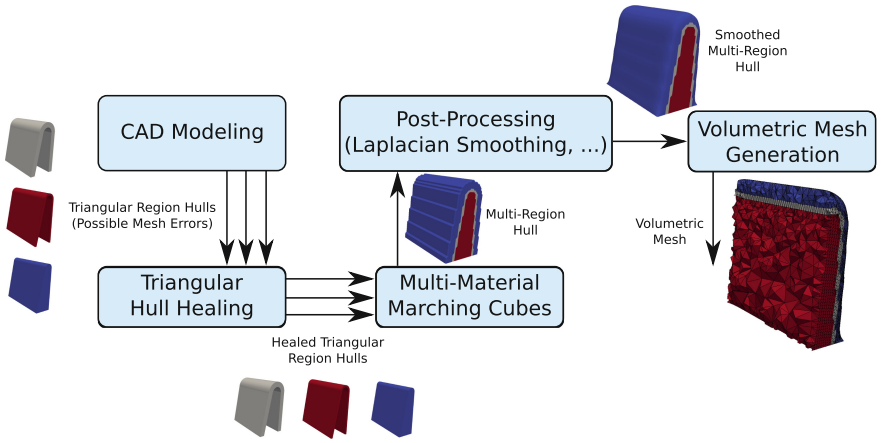

Fig. 2. Each region of a device (represented by one arrow) is exported individually by the CAD tools to a triangular hull and *healed* (cf. Sect. 3.1). A multi-material marching cubes algorithm re-samples these *healed* triangular hulls (cf. Sect. 3.2) and creates a valid multi-region hull. After a post-processing step, a volumetric mesh is generated based on the re-sampled geometry (cf. Sect. 3.3).

healing tools, implementing some of these algorithms, are freely availably [16]. However, most of the algorithms originate in the field of computer graphics and are therefore not able to handle multiple regions properly, which is highly relevant for the field of TCAD.

3 Mesh Healing and Generation Pipeline

In this section, a mesh *healing* and generation pipeline is presented, an overview of which is given in Fig. 2. The pipeline consists of three parts as described in the following subsections.

3.1 CAD Interface and Triangular Hull Healing

After modeling the device in a CAD tool, each region is exported on its own using a triangular hull representation. These triangular region hulls will only be used in the re-sampling step to test if a point is inside the region hull. Therefore, region interfaces do not need to be compatible and each region hull can be treated individually. To ensure stable point inclusion tests, triangular hull errors are *healed* using the mesh *healing* tool Polymender [17].

3.2 Re-Sampling

After *healing* the region hulls, a volumetric re-sampling is performed by creating a regular three-dimensional grid covering the entire device geometry.

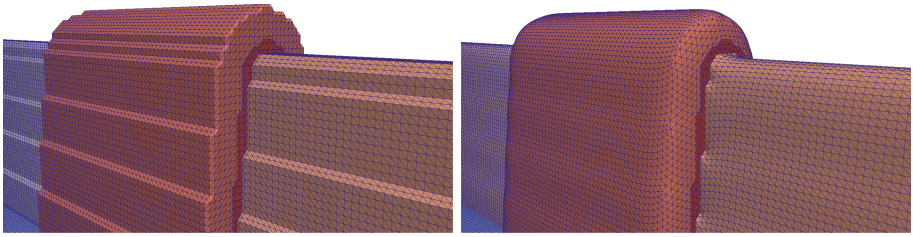

Fig. 3. A re-sampled triangular hull geometry before and after the smoothing process.

For each grid point the corresponding region is determined by using a point-in-hull test on each *healed* region hull. Due to possibly different geometry discretizations at interfaces, regions might intersect or form holes. If a grid point is in more than one mesh region, a user-created priority list resolves the ambiguity. Afterwards, a three-dimensional version of a dilatation and an erosion operation avoids holes in the re-sampled geometry [18]. This regular grid is then used by the multi-material marching cubes algorithm [19] in order to create a triangular hull with multiple regions and valid region interfaces. The entire re-sampling step has been implemented in ViennaMesh.

3.3 Post Processing and Volumetric Mesh Generation

Due to the nature of the marching cubes algorithm, the re-sampled triangular hull has a stair-stepped characteristic. A modified version of the Laplacian smoothing algorithm is applied to mitigate these characteristics [19]. Figure 3 visualizes the re-sampled triangular hull before and after Laplacian smoothing. Like the re-sampling step, the Laplacian smoothing algorithm has also been implemented in ViennaMesh. Depending on the application, chosen grid resolution during the re-sampling step, and required volumetric mesh resolution, a refinement or coarsening algorithm suitable for multi-region triangular hulls is applied on the smoothed mesh. Finally, a mesh generation software, like Tetgen, is used to create a volumetric ready-to-simulate mesh based on the resulting triangular hull.

4 Examples

In this section we apply our algorithm to a bulk silicon trigate transistor [20] and a FlexFET [21], which have been modeled with the free open source CAD tool FreeCAD.

The exported geometry of the bulk silicon trigate transistor has 22 volumetric holes and 24 intersections, visualized in Fig. 4. By applying our algorithm pipeline, we obtain a valid multi-region triangular hull geometry, where all errors of the exported input geometry have been successfully eliminated.

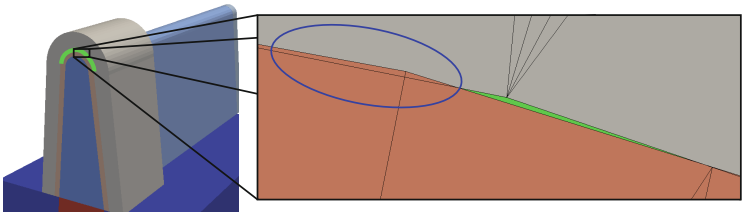

Fig. 4. Hole (marked green) and intersection errors (marked blue) in the bulk silicon trigate transistor due to different discretizations of neighboring regions. The green area on the left indicates the areas, where these errors occur (color figure online).

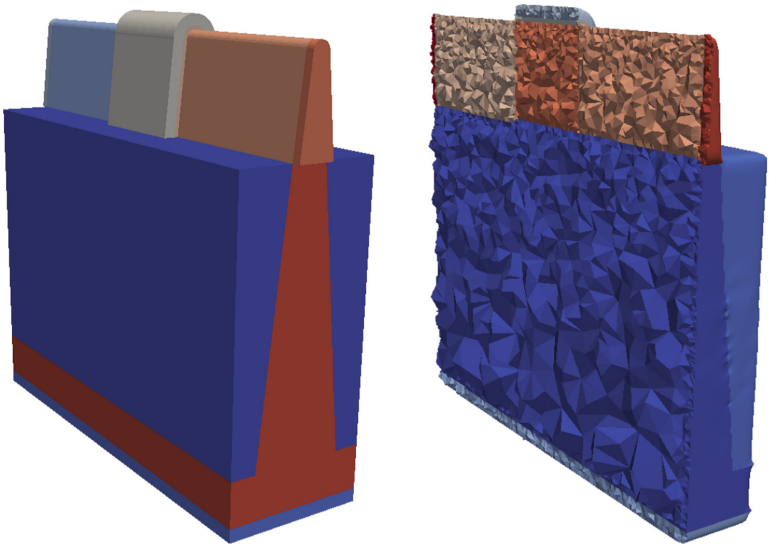

Fig. 5. The bulk silicon trigate transistor: The original geometry modeled in FreeCAD (left) and a clipped visualization of the generated volumetric mesh (right).

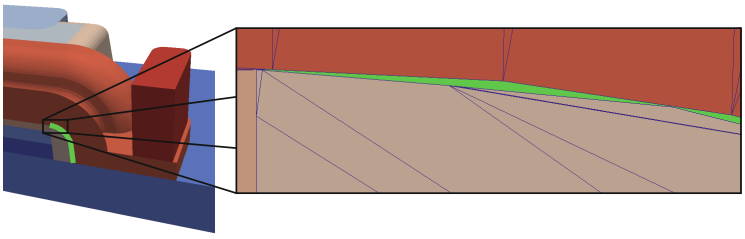

Fig. 6. A hole in the exported FlexFET geometry (visualized in green) which stems from modeling inaccuracies (color figure online).

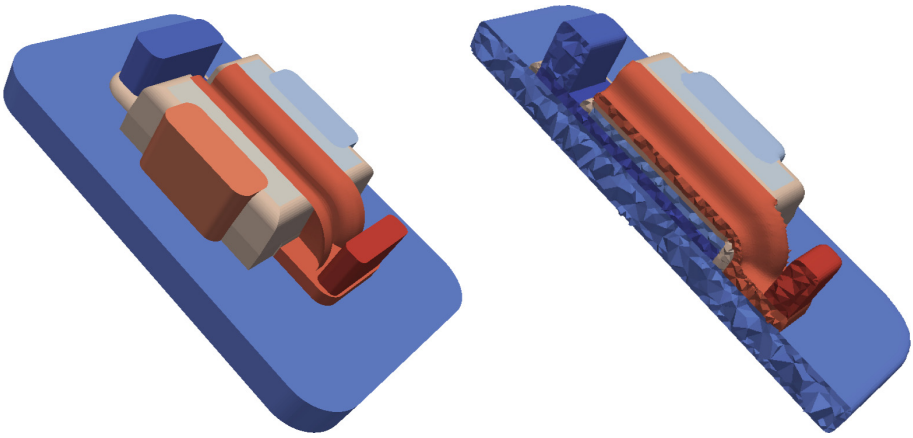

Fig. 7. The FlexFET: The original geometry modeled in FreeCAD (left) and a clipped visualization of the generated volumetric mesh (right).

This *healed* triangular hull geometry is used by ViennaMesh's Tetgen module to create a volumetric ready-to-simulate mesh. The modeled device geometry and the volumetric mesh are visualized in Fig. 5.

The exported FlexFET geometry has a total of 39 errors, being 21 volumetric holes and 18 intersections. In contrast to holes caused by different surface discretizations, the FlexFET geometry has a hole which stems from modeling inaccuracies (cf. Fig. 6). Again, applying our algorithm pipeline successfully *heals* all errors and generates a valid multi-region triangular hull geometry. If, in case of a high re-sampling resolution, holes are not closed, the kernel size of the three-dimensional dilatation and erosion operation during the re-sampling step has to be increased. ViennaMesh's Tetgen module is used to create a volumetric ready-to-simulate mesh based on the *healed* triangular hull. The modeled FlexFET geometry and the volumetric mesh is shown in Fig. 7.

5 Summary and Future Work

We presented an algorithm pipeline for automatic and robust *healing* of CAD geometries for further processing by volumetric mesh generation tools. In contrast to mesh and geometry *healing* algorithms used in the field of computer graphics, our approach supports meshes with multiple regions. We show, that our algorithm pipeline reliably generates volumetric ready-to-simulate meshes based on geometries of complex semiconductor devices modeled in CAD tools.

To further improve the stability for handling big holes which are not closed by dilatation and erosion operations, other filling algorithms, like the flood fill algorithm [18], should be investigated in the future.

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

References

1. AutoCAD: http://www.autodesk.de/products/autocad/overview/
2. FreeCAD: http://www.freecadweb.org/
3. Synopsys Sentaurus Structure Editor: http://www.synopsys.com/Tools/TCAD/Pages/StructureEditor.aspx
4. DEVSIM: https://github.com/devsim/devsim/
5. Strikwerda, J.C.: Finite difference schemes and partial differential equations, 2nd edn. SIAM, Philadelphia (2004) ISBN: 978-0-89871-567-5
6. Cheng, S.W., Dey, T.K., Shewchuk, J.R.: Delaunay Mesh Generation. CRC Press, Boca Raton (2013) ISBN: 978-1584887300
7. Pratt, M.J.: Introduction to ISO 10303 - the STEP standard for product data exchange. J. Comput. Inf. Sci. Eng. **1**(1), 102–103 (2001). doi:10.1115/1.1354995
8. Si, H.: TetGen a quality tetrahedral mesh generator and three-dimensional delaunay triangulator, Version 1.4, User Manual (2006). http://wias-berlin.de/software/tetgen/files/tetgen-manual.pdf
9. Szilvási-Nagy, M., Mátyási, G.: Analysis of STL files. J. Math. Comput. Model. **38**(7–9), 945–960 (2003). doi:10.1016/S0895-7177(03)90079-3
10. ViennaMesh: http://viennamesh.sourceforge.net/
11. Attene, M., Campen, M., Kobbelt, L.: Polygon mesh repairing: an application perspective. ACM Comput. Surv. **45**(2), 1–33 (2013). doi:10.1145/2431211.2431214
12. Chong, C., Kumar, A.S., Lee, H.: Automatic mesh-healing technique for model repair and finite element model generation. J. Finite Elem. Anal. Des. **43**(15), 1109–1119 (2007). doi:10.1016/j.finel.2007.06.009
13. Frederick, C., Wong, Y., Edge, F.: Two-dimensional automatic mesh generation for structural analysis. Int. J. Numer. Meth. Eng. **2**, 133–144 (1970). doi:10.1002/nme.1620020112
14. Hoppe, H.: Progressive meshes. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 99–108. New York (1996). doi:10.1145/237170.237216
15. Ju, T.: Robust repair of polygonal models. ACM Trans. Graph. **23**(3), 888–895 (2004). doi:10.1145/1015706.1015815
16. Mesh Repairing Software on the Web: http://meshrepair.org/
17. Polymender: http://www1.cse.wustl.edu/taoju/code/polymender.htm
18. Burger, W., Burge, M.J.: Digital Image Processing - An Algorithmic Introduction Using Java. Texts in Computer Science, 1st edn. Springer-Verlag, London (2008)
19. Wu, Z., Sullivan, J.M.: Multiple material marching cubes algorithm. Int. J. Numer. Meth. Eng. **58**(2), 189–207 (2003). doi:10.1002/nme.775
20. Agrawal, N., Kimura, Y., Arghavani, R., Datta, S.: Impact of transistor architecture (bulk planar, trigate on bulk, ultrathin-body planar SOI) and material (silicon or III-V semiconductor) on variation for logic and SRAM applications. IEEE Trans. electron devices **60**(10), 3298–3304 (2013). doi:10.1109/TED.2013.2277872
21. Modzelewski, K., Chintala, R., Moolamalla, H., Parke, S., Hackler, D.: Design of a 32nm independently-double-gated FlexFET SOI transistor. In: Proceedings of the 17th Biennial University/Government/Industry Micro/Nano Symposium, region hulls, a volumetric pp. 64–67 (2008) doi:10.1109/UGIM.2008.24