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/© 2015 Elsevier Inc. All rights reserved.

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