ViennaMesh
A Highly Flexible Meshing Framework

Florian Rudolf, Karl Rupp, and Siegfried Selberherr

Institute for Microelectronics
Technische Universität Wien
Vienna - Austria - Europe
http://www.iue.tuwien.ac.at

Table of Contents

1 Introduction

2 The Framework

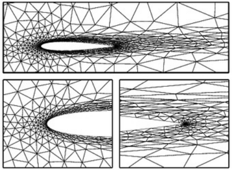
3 Use Case

4 Conclusion

Motivation

Different applications require different meshing properties

- Mesh quality is context-dependent



Different mesher generate meshes with different properties

- Most meshing algorithms are only suitable for a specific set of applications

Simultaneous use of different meshing algorithms/libraries

- Challenging due to incompatible data structures and interfaces

Existing Frameworks and Libraries

Often only one algorithm type is implemented

- Usage is restricted to certain set of applications

Static data structure

- Conversion from and to other formats is hard or even impossible

Static interface

- Extensibility is hard or even impossible

Our Approach: ViennaMesh

Open source C++ meshing framework

- Multi-segment support

Based on a highly flexible data structure

- Focus on abstract topology

Abstract concepts enable uniform interfaces

- Generic programming

Library Details

C++ library

- Data structure and core functionality header-only
- C++11 support

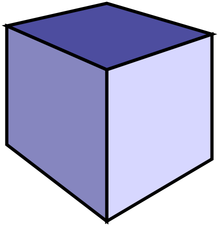
Interfaces to proven libraries

- Meshing: Netgen, CGAL, VERDICT
- Statistics: Boost.Accumulators
- More to come: Tetgen, Triangle, Mesquite, ...

ViennaMesh - Topology

Data structure is topology-driven

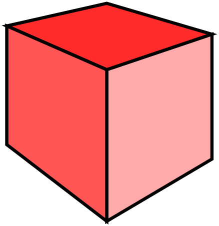
- Abstract topology concept
- **Element, cell**, boundary element, co-boundary element



ViennaMesh - Topology

Data structure is topology-driven

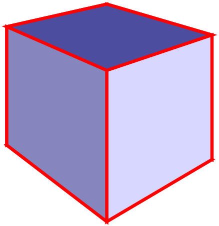
- Abstract topology concept
- Element, cell, **boundary element**, co-boundary element



ViennaMesh - Topology

Data structure is topology-driven

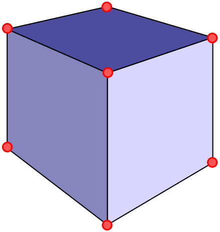
- Abstract topology concept
- Element, cell, **boundary element**, co-boundary element



ViennaMesh - Topology

Data structure is topology-driven

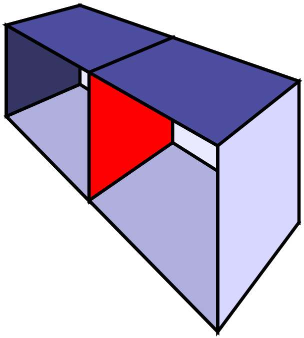
- Abstract topology concept
- Element, cell, **boundary element**, co-boundary element



ViennaMesh - Topology

Data structure is topology-driven

- Abstract topology concept
- Element, cell, boundary element, **co-boundary element**



ViennaMesh - Data Structure

Highly flexible topological structure

- Simplices and hypercubes of arbitrary topological dimension
- Polygons and PLCs

Compile-time configuration using C++ templates

- Better performance than run-time data structures

ViennaMesh - Data Structure

Topological complex

- Set of elements
- Intersections of 2 elements $\rightarrow$ empty or another element

Topology and geometry are independent $\rightarrow$ separation

- e.g. triangles in 3D space

Iterator interface is type-independent

- Many algorithms can be written type-independently, e.g. data transfer
- Optimum: one source code for all element types

ViennaMesh Data Structure Example - Data transfer

Transfer data from triangle to vertex

- value weighted with triangle volume

```
for (auto v : viennagrid::vertices( domain ) )
{
  numeric_type weighted_value = 0, total_volume = 0;

  for ( auto t : viennagrid::triangles(domain, v) )
  {
    numeric_type current_volume = volume( domain, t );
    total_volume += current_volume;
    weighted_value += current_volume * value(t);
  }

  value(v) = weighted_value / total_volume;
}
```

ViennaMesh Data Structure Example - Data transfer

Type independent implementation

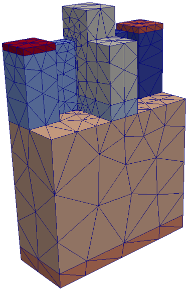
- to_tag and from_tag specify the types

```
for (auto v :  viennagrid::elements<to_tag>( domain ) )
{
  numeric_type weighted_value = 0, total_volume = 0;

  for ( auto t :  viennagrid::coboundary_elements<from_tag>(domain, v) )
  {
    numeric_type current_volume = volume( domain , t );
    total_volume += current_volume;
    weighted_value += current_volume * value(t);
  }

  value(v) = weighted_value / total_volume;
}
```

ViennaMesh - Segment Support

Support for segments

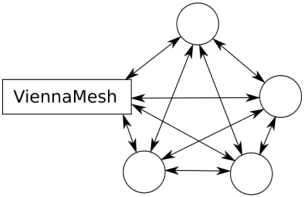
- Subsets of the mesh
- Preserve interfaces through meshing process



ViennaMesh - Domain Concept

External library support

- External data structures



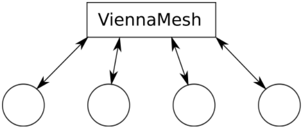
ViennaMesh - Domain Concept

External library support

- External data structures

Only 2 conversions per external data structure needed

- From and to ViennaMesh



ViennaMesh - Domain Concept

Much less conversion functions needed

$\rightarrow$ $O(N)$ conversions instead of $O(N^2)$

Conversion time about 1% of the algorithm time

$\rightarrow$ no performance killer

High configurability of topology

$\rightarrow$ ensures compatibility with other meshing data structures

ViennaMesh - Supported Algorithms

Externally provided algorithms

- Netgen: Triangular hull $\rightarrow$ Tetrahedron, multi-segment support
- CGAL: PLC $\rightarrow$ Triangular hull
- CGAL: Triangular hull $\rightarrow$ Tetrahedron

Internally provided algorithms

- Multi-segment hull refinement
- Seed point segmenting
- Hull/Geometry extraction

ViennaMesh - Meshing Algorithms

External and internal algorithms share common interface

- data structure conversion if needed

```
InputDomainType  input_domain;
OutputDomainType output_domain;
viennamesh::result_of::settings<algorithm_tag>::type
                 settings;

settings.cell_size = 1.0;

viennamesh::run_algo< algorithm_tag >(
    input_domain,
    output_domain,
    settings);
```

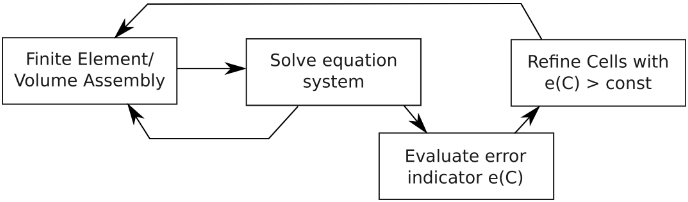
ViennaMesh - Meshing Algorithms

External and internal algorithms share common interface

- Easy exchangeability of algorithms

```
InputDomainType  input_domain;
OutputDomainType output_domain;
viennamesh::result_of::settings< algorithm_tag >::type
                settings;

settings.cell_size = 1.0;

viennamesh::run_algo< algorithm_tag >(
    input_domain,
    output_domain,
    settings);
```

Use Case - Adaptive Finite Element/Volume Method



Adaptive Loop

- Error indicator $e(C) := max_{A \in \text{neighbour}(C)} \{|\text{value}(A) - \text{value}(C)|\}$
- Cells with $e(C) > \text{const}$ are refined

Use Case - Drift Diffusion on 3D MOSFET Transistor

Drift Diffusion

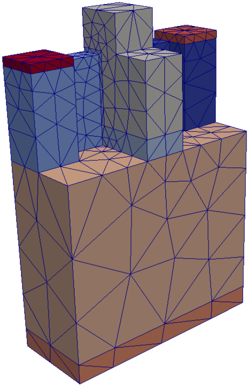
$$\begin{aligned} \nabla \cdot (\epsilon \nabla \psi) &= q((n - N_D) - (p - N_A)) \\ \nabla \cdot (D \nabla n - \mu n \nabla \psi) &= 0 \\ \nabla \cdot (D \nabla p + \mu p \nabla \psi) &= 0 \end{aligned}$$

Drift Diffusion on 3D MOSFET transistor

- Using Finite Element/Volume assembly
- Starting mesh: 3200 cells

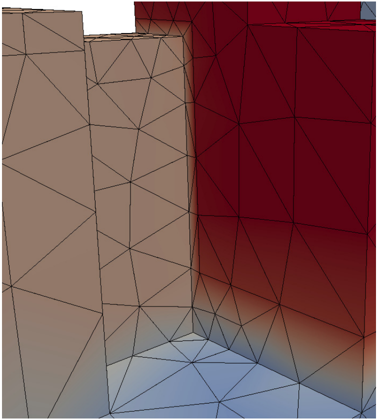
J J H Miller et al, Report on Progress in Physics, 1999

Drift Diffusion on 3D MOSFET Transistor

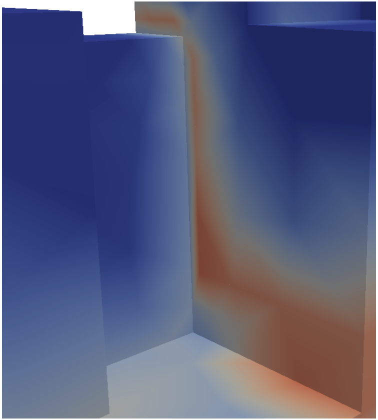


Initial mesh with 3200 cells

Refinement - First Iteration

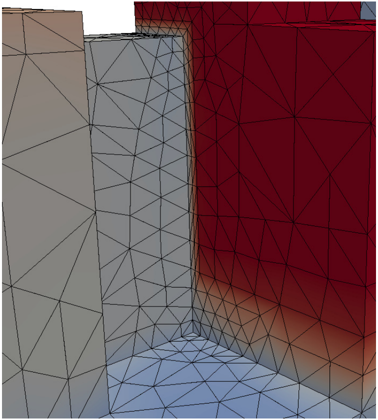


Potential

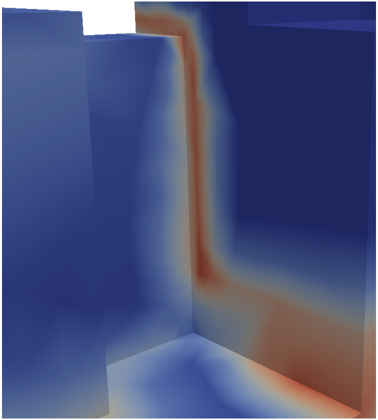


Error indicator

Refinement - Second Iteration

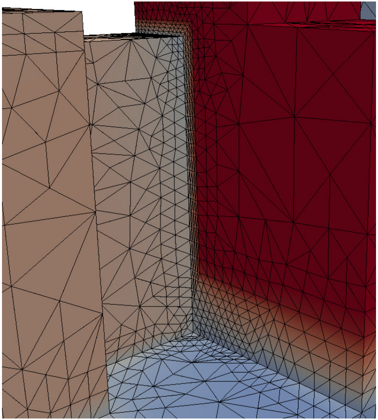


Potential

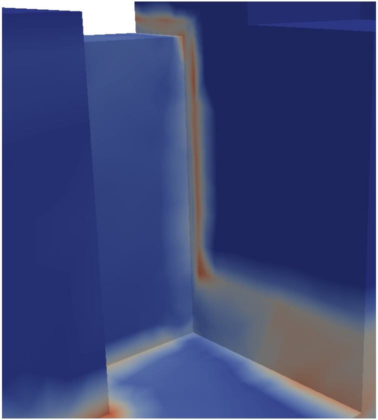


Error indicator

Refinement - Third Iteration

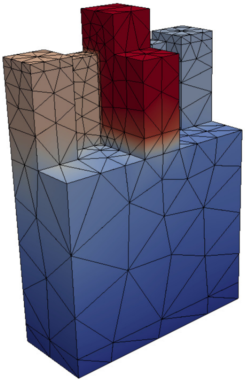


Potential

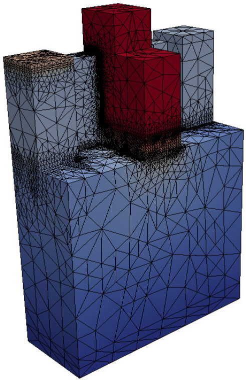


Error indicator

Drift Diffusion on 3D MOSFET Transistor



Initial mesh with 3200 cells



Iteration 4 with 467813 cells

Use Case - Drift Diffusion on 3D MOSFET Transistor

Cell count

iteration	total cells	bad cells	relative bad cells
1	3 200	2 082	65.06%
2	20 731	8 270	39.89%
3	103 366	40 302	38.99%
4	467 813	63 956	13.67%
5	1 349 050	15 144	1.12%
6	1 727 444	6 194	0.36%

Use Case - Drift Diffusion on 3D MOSFET Transistor

Cell count

iteration	total cells	bad cells	relative bad cells
1	**3 200**	**2 082**	**65.06%**
2	20 731	8 270	39.89%
3	103 366	40 302	38.99%
4	467 813	63 956	13.67%
5	1 349 050	15 144	1.12%
6	**1 727 444**	**6 194**	**0.36%**

Conclusion

Flexibility

- Common interface $\rightarrow$ Easily change meshing kernel
- Abstract concepts $\rightarrow$ Write your code only once
- High configurability $\rightarrow$ Use arbitrary topological structures
- High extensibility $\rightarrow$ Write your own meshing algorithm

Status

- Development release available at sourceforge
- http://viennamesh.sourceforge.net