Sentaurus™ Mesh User Guide

Version T-2022.03, March 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

Contents

About This Guide

The Synopsys® Sentaurus™ Mesh tool is a mesh generator that incorporates different mesh generation engines: an axis-aligned mesh generator, an offsetting mesh generator, and a tensor-product mesh generator that produces rectangular or hexahedral elements.

Sentaurus Mesh is designed to be used in a wide range of simulators, including the Synopsys TCAD products Sentaurus Device, Sentaurus Process, Sentaurus Device Electromagnetic Wave Solver, and Sentaurus Interconnect. Local mesh refinement is performed using the doping and refinement information in the mesh command file.

For additional information, see:

- The TCAD Sentaurus release notes, available on the Synopsys SolvNetPlus support site (see Accessing SolvNetPlus on page 9)
- Documentation available on the SolvNetPlus support site

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field or an option.
`Courier font`	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.
Italicized text	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.

Customer Support

Customer support is available through the Synopsys SolvNetPlus support site and by contacting the Synopsys support center.

Accessing SolvNetPlus

The SolvNetPlus support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNetPlus site:

1. Go to https://solvnetplus.synopsys.com.
2. Enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys Global Support Centers site on www.synopsys.com. There you can find email addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNetPlus site or the Synopsys Global Support Centers site and open a case (Synopsys user name and password required).

Contacting Your Local TCAD Support Team Directly

Send an email message to:

- support-tcad-us@synopsys.com from within North America and South America
- support-tcad-eu@synopsys.com from within Europe
- support-tcad-ap@synopsys.com from within Asia Pacific (China, Taiwan, Singapore, Malaysia, India, Australia)
- support-tcad-kr@synopsys.com from Korea
- support-tcad-jp@synopsys.com from Japan

1

Introduction to Sentaurus Mesh

This chapter describes how to start Sentaurus Mesh and provides a general explanation of its functionality.

Functionality of Sentaurus Mesh

Sentaurus Mesh is a suite of tools that produce finite-element meshes for use in applications such as semiconductor device simulations, process simulations, and electromagnetic simulations. It has three mesh generation engines: an axis-aligned mesh generator, an offsetting mesh generator, and a tensor-product mesh generator. Sentaurus Mesh also provides a set of tools that perform operations on boundary representations and meshes.

The axis-aligned and offsetting mesh generators produce Delaunay meshes, which are suitable for use in Sentaurus Device and Sentaurus Process. In one dimension, the meshes contain segments only. In two dimensions, the meshes contain triangles only, while in three dimensions, the meshes comprise tetrahedra. For information about the algorithm used to generate Delaunay meshes, see Chapter 6 on page 146.

The offsetting mesh generator can produce layered meshes in two and three dimensions. The layers are located at the device interfaces and follow the contours of the interface. They can be combined with axis-aligned elements to produce high-quality meshes for nonplanar structures. As such, the offsetting mesh generator is a superset of the axis-aligned mesh generator, where layering takes precedence over axis-aligned mesh generation.

The tensor-product mesh generator is intended to generate meshes for Sentaurus Device Electromagnetic Wave Solver and for some applications in Sentaurus Device. The meshes contain rectangular elements in two dimensions and cuboid elements in three dimensions.

Sentaurus Mesh reads the input geometry from a boundary file stored in the TDR format with the `_bnd.tdr` file extension. Some TDR files from Sentaurus Process and Sentaurus Interconnect with the `_fps.tdr` and `_sis.tdr` file extensions, respectively, contain two geometry objects: one for the volumetric data and one for the boundary representation. Sentaurus Mesh reads the boundary object in these TDR files, but it ignores other geometry objects.

Impurity concentrations and user-required element sizes can be described using a mesh command file. The grid can be adapted to analytic profiles generated by Sentaurus Structure Editor or profiles generated by Sentaurus Process. (All references to concentrations in this document imply ‘active’ or ‘substitutional’ concentrations, since calculations in Sentaurus Device use concentrations in this form.)

The required point density is obtained by refining the elements in an anisotropic way. Therefore, unnecessary point propagation due to quadtrees, octrees, or tensor-product grid techniques is avoided.

A delaunization process allows Sentaurus Mesh to obtain high-quality conforming Delaunay grids, suitable for control volume discretization methods that are used in device simulation. For more information, refer to the literature [1][2][3][4][5].

The output of Sentaurus Mesh depends on the mesh generation engine used. The axis-aligned and offsetting mesh generators always produce a TDR unstructured mesh; the tensor-product mesh generator will select the type of mesh depending on the target application.

Applications of Different Mesh Generators

The choice of which mesh generator to use for a particular application depends largely on the geometry of the device.

For devices where the most important surfaces are axis aligned, the recommendation is to use the axis-aligned mesh generator, since it produces the highest quality elements with minimal node count for such devices.

For devices where the main surfaces are nonaxis-aligned or curved (for example, a MOS-type structure where the channel is nonplanar), the recommendation is to use the offsetting mesh generator, since it produce meshes containing layers that better conform to the curved surfaces, thereby reducing the number of elements in the final mesh (see Offsetting Section on page 47).

For electromagnetic simulations using Sentaurus Device Electromagnetic Wave Solver, use the tensor-product mesh generator.

Starting Sentaurus Mesh

In Sentaurus Mesh, a mesh is created from two input files, namely, the boundary file and the command file. If the input project is called `project_name`, a mesh can be created using the command:

```
snmesh [<options>] project_name
```

Sentaurus Mesh automatically adds the extensions `_bnd.tdr` and `.cmd` to the base name `project_name`. It also creates the output file `project_name_msh.tdr` that contains mesh geometry information and doping information. Another file, `project_name_msh.log`, is created and is used as the log file for the mesh generation.

Command-Line Options

The binary of Sentaurus Mesh is `snmesh`. It is executed using the syntax:

```
snmesh [<options>] <command_file_name>
```

Table 1 Command-line options available for Sentaurus Mesh

Option	Description
`-backcompat <release>`	Changes the behavior of the mesh generation and solid-modeling algorithms and variable defaults to those of a specified previous release. For example: `-backcompat S-2021.06`
`-h`	Displays help information.
`-H`	Activates hybrid mesh generation.
`--max_threads <integer>`	Specifies an upper limit for the number of threads used by the mesh generators (axis-aligned, offsetting, and tensor-product). This parameter overrides any values for the number of threads specified in the command file.
`--threads <integer>`	Specifies the global number of threads used by the mesh generators (axis-aligned, offsetting, and tensor-product). This value overrides the `numThreads` value specified in the `IOControls` section.
`-v`	Displays version information only.

TCAD Sentaurus Tutorial: Simulation Projects

The TCAD Sentaurus Tutorial provides various projects demonstrating the capabilities of Sentaurus Mesh.

To access the TCAD Sentaurus Tutorial:

1. Open Sentaurus Workbench by entering the following on the command line: `swb`
2. From the menu bar of Sentaurus Workbench, choose **Help** > **Training** or click  on the toolbar.

Alternatively, to access the TCAD Sentaurus Tutorial:

1. Go to the `$STROOT/tcad/current/Sentaurus_Training` directory.

   The `STROOT` environment variable indicates where the Synopsys TCAD distribution has been installed.

2. Open the `index.html` file in your browser.

References

[1] L. Villablanca, *Mesh Generation Algorithms for Three-Dimensional Semiconductor Process Simulation*, Series in Microelectronics, vol. 97, Konstanz, Germany: Hartung-Gorre, 2000.

[2] P. Conti, M. Tomizawa, and A. Yoshii, “Generation of Oriented Three-Dimensional Delaunay Grids Suitable for the Control Volume Integration Method,” *International Journal for Numerical Methods in Engineering*, vol. 37, no. 19, pp. 3211–3227, 1994.

[3] G. Garretón *et al*., “A New Approach for 2-D Mesh Generation for Complex Device Structures,” in *International Workshop on Numerical Modeling of Processes and Devices for Integrated Circuits (NUPAD V)*, Honolulu, HI, USA, pp. 159–162, June 1994.

[4] G. Garretón *et al*., “Unified Grid Generation and Adaptation for Device Simulation,” in *Simulation of Semiconductor Devices and Processes (SISDEP)*, vol. 6, Erlangen, Germany, pp. 468–471, September 1995.

[5] G. Heiser, *Design and Implementation of a Three-Dimensional General Purpose Semiconductor Device Simulator*, Series in Microelectronics, vol. 13, Konstanz, Germany: Hartung-Gorre, 1991.

2

Command File

This chapter describes the sections of the command file of Sentaurus Mesh.

Basics of the Command File

In the command file (`.cmd`), you can specify different parameters for the generation of a mesh as follows:

- Sections are delimited by opening and closing braces.
- Only one keyword must be specified per line.
- Keywords used in the command file are not case sensitive.
- Strings are enclosed in double quotation marks.
- Comments start with `*` or `#`.

Different types of information can be given in the command file. You can specify refinement information, doping profile information, and control parameters for the different mesh generators and tools provided in Sentaurus Mesh.

Refinement information is required to control mesh generation according to user requirements (local element size). This information is specified in the `Definitions` section. Profile information is required to define the fields, for example, doping profiles, which are used in grid adaptation. Doping profiles can be specified with different types of information:

- External simulation results
- Constant data
- Analytic formulas and predefined functions describing a profile

Sections of the Command File

The command file has the following sections:

- The command file can start with an optional title statement, which consists of the `Title` keyword followed by a string in double quotation marks. By default, `Title ""` is used.
- `IOControls` specifies an explicit input file containing the structure and an output file to which the generated mesh will be saved (see IOControls Section on page 16).
- `Definitions` defines the sets of refinement parameters and profile definitions to be used in the `Placements` section. These sets are referred to using their unique *reference name* (see Definitions Section on page 18).
- `Placements` defines instances of the definitions given in the `Definitions` section, placed with respect to the current device (see Placements Section on page 31).
- `Interpolate` controls data interpolation (see Interpolate Section on page 41).
- `AxisAligned` controls the axis-aligned mesh generator (see AxisAligned Section on page 42).
- `Offsetting` controls the offsetting mesh generator (see Offsetting Section on page 47).
- `Delaunizer` controls the behavior of the delaunizer (see Delaunizer Section on page 49).
- `Tensor` controls the tensor-product mesh generator (see Tensor Section on page 53).
- `Boundary` controls the parameters related to boundary algorithms (see Boundary Section on page 63).
- `Tools` specifies additional utilities (see Tools Section on page 68).
- `QualityReport` specifies the mesh quality statistics to be reported and the limits for the mesh quality criteria (see QualityReport Section on page 89).
- `Structured` specifies the parameters required to define a Josephson junction mesh (see Structured Section on page 92).

The syntax of the command file is:

```
Title ""
IOControls {input/output information}
Definitions {defining information}
Placements {placing information}
Interpolate {data interpolation information}
AxisAligned {axis alignment information}
Offsetting {offsetting information}
Delaunizer {delaunizer information}
Tensor {tensor information}
Boundary {boundary information}
```

```
Tools {tools information}
QualityReport {mesh quality information}
Structured {Josephson junction mesh information}
```

IOControls Section

This section is used to specify the names of input files describing the structure and the name of the output file with the generated result. Input and boundary files can contain either a boundary or a mesh in TDR format.

You can use the `EnableOffset`, `EnableSections`, `EnableTensor`, and `EnableTools` options to enable different algorithms based on the contents of the command file. The result after activating unrelated sections in the command file is undefined.

Syntax of IOControls Section

```
IOControls {
   EnableEMW
   EnableIsotropic
   EnableOffset
   EnableSections
   EnableTensor
   EnableTools
   inputFile = "string"
   numThreads = integer
   outputFile = "string"
   saveInterfaceRegions = true | false
   unstructuredTensorMesh = true | false
   useDFISEcoordinates
   useUCScoordinates
   verbosity = 0 | 1 | 2 | 3
}
```

Parameters

Default values are given in parentheses if applicable.

`EnableEMW`

Generates meshes suitable for Sentaurus Device Electromagnetic Wave Solver (EMW) applications using the tensor-product mesh generator (see Computing Cell Size Automatically on page 57).

`EnableIsotropic`

Activates the isotropic mesh generator, which performs refinement directly on the elements instead of axis-aligned spatial refinement. The delaunization parameters for an isotropic mesh are controlled by the settings in the `Delaunizer` section of the command file (see Delaunizer Section on page 49).

`EnableOffset`

Activates the `Offsetting` section of the command file and the offsetting mesh generator (see Offsetting Section on page 47).

`EnableSections`

Parses the command file and activates the mesh generators associated with the sections present in the command file. If the command file contains the `AxisAligned`, `Offsetting`, `Tensor`, or `Tools` sections, the `EnableSections` option activates automatically the corresponding mesh generators.

Specifying `EnableSections` enables the following default behavior:

- If none of the `Offsetting`, `Tensor`, and `Tools` sections are present in the command file, then the axis-aligned mesh generator is selected.
- If either the `Tensor` section or the `Tools` section is present in the command file, then that section has priority over the `Offsetting` section, which is ignored if it is present.
- If either the `Tensor` section or the `Tools` section is present in the command file, then the axis-aligned mesh generator is ignored.
- If both the `Tensor` and `Tools` sections are present in the command file, then the `Tensor` section has priority over the `Tools` section.

Note:
The `Definitions` and `Placements` sections belong to the analytic profile generator and can be executed only by the axis-aligned mesh generator, the tensor-product mesh generator, or the `CreateProfiles` subsection of the `Tools` section. The `Definitions` and `Placements` sections are ignored if any other tool or module is executed.

`EnableTensor`

Activates the tensor-product mesh generator (see Tensor Section on page 53).

`EnableTools`

Activates the operations specified in the `Tools` section (see Tools Section on page 68).

`inputFile`

The name of the default input file is based on the name of the command file. If an input file is specified, then it is used instead of the default input file based on the name of the command file.

`numThreads (1)`

Sets the number of threads to be used by the mesh generators (axis-aligned, offsetting, and tensor-product). This value can be overridden by using the `--max_threads` and `--threads` command-line options.

`outputFile`

Specifies the name of the output file.

`saveInterfaceRegions (true)`

Specifies whether Sentaurus Mesh generates interface regions in the mesh.

`unstructuredTensorMesh (false)`

Specifies whether Sentaurus Mesh creates a tensor mesh that is compatible with Sentaurus Device.

`useDFISEcoordinates`

Converts all coordinates to the DF–ISE coordinate system (except the coordinates from the command file).

`useUCScoordinates`

Uses the unified coordinate system (UCS). With the exception of the command file, the coordinates from all files read by Sentaurus Mesh are converted to the UCS.

`verbosity`

Sets the verbosity level of output messages. With `verbosity=0`, only basic messages are displayed. With `verbosity=3`, all messages are displayed.

Definitions Section

This section consists of sets of refinement and profile subsections. Each subsection consists of a reference name, an opening brace, the specification of parameters, and a closing brace.

The order of definitions in the `Definitions` section is not important since these definitions are used as references in the `Placements` section.

Syntax of Definitions Section

```
Definitions {
   AnalyticalProfile "reference name" {parameters}
   Constant "reference name" {parameters}
   Multibox "reference name" {parameters}
   Particle "reference name" {parameters}
   Refinement "reference name" {parameters}
   SubMesh "reference name" {parameters}
   ...
}
```

Defining Analytic Profiles

You can define profiles by using simple analytic expressions, which have two components. The first component `Function` represents the values along a direction defined as the normal direction of the `ReferenceElement`. This is the *primary direction*. These values are smoothed along the direction perpendicular to the normal, or *lateral direction*, using the second component `LateralFunction`.

These expressions can be of the following types:

- Predefined functions: Gaussian and error function
- One-dimensional external profile
- Your own function (by using the general function evaluator)

The formulas used for these analytic profiles are described in Appendix A on page 150.

To define an analytic profile (instead of `AnalyticalProfile`, you can use its abbreviation `AnaProf`), use the following syntax:

```
Definitions {
   AnalyticalProfile "reference name" {
      Species = "string"
      Function = Gauss(primary parameters) | Erf(primary parameters) |
                 subMesh1D(primary parameters) |
                 Eval(primary parameters) | General(primary parameters)
      LateralFunction = Gauss(lateral parameters) |
                        Erf(lateral parameters) |
                        Eval(lateral parameters)
   }
}
```

Parameters

`Species`

Specifies the species for the analytic profile.

`Function`

Indicates the type of function and the parameters used along the primary direction, the direction normal to the `ReferenceElement`.

`LateralFunction`

Defines the lateral component of the analytic profile (you also can use its abbreviation `LatFunc`). A Gaussian function, an error function, or a general analytic function can be specified using the following lateral parameters:

```
LateralFunction = Gauss(Factor = float)
LateralFunction = Gauss(StandardDeviation = value)
LateralFunction = Gauss(Length = value)
```

```
LatFunc = Erf(Factor = float)
LatFunc = Erf(Length = value)
LatFunc = Eval(init = "..." function = "...")
```

Note:

The default `LateralFunction` is the error function `Erf`.

If you use `General` to specify the analytic profile, then there is no separate `LateralFunction` since the definition of the analytic profile using `General` includes both the primary and the lateral directions in its formulation.

Specifying a Gaussian Function

A Gaussian function can be specified with the following primary parameters:

```
Function = Gauss(PeakPosition=value, PeakValue=value,
           StandardDeviation=value)
Function = Gauss(PeakPosition=value, Dose=value, StdDev=value)
Function = Gauss(PeakPosition=value, PeakValue=value, Length=value)
Function = Gauss(PeakPosition=value, Dose=value, Length=value)
Function = Gauss(PeakPosition=value, PeakValue=value,
           ValueAtDepth=value, Depth=value)
Function = Gauss(PeakPos=value, Dose=value, ValueAtDepth=value,
           Depth=value)
```

By default, `PeakPosition=0`. There are no default values for the other parameters.

When `Function=Gauss`, `Factor=0.8` in `LateralFunction` by default.

Some parameters have abbreviations (in parentheses) you can use, such as:

- `PeakPosition` (`PeakPos`)
- `PeakValue` (`PeakVal`)
- `StandardDeviation` (`StdDev`)
- `ValueAtDepth` (`ValAtDepth`)

Specifying an Error Function

An error function can be specified with the following primary parameters:

```
Function = Erf(SymmetryPosition = value, MaxValue = value,
           Length = value)
Function = Erf(SymmetryPosition = value, Dose = value, Length = value)
Function = Erf(SymPos = value, MaxVal = value, ValueAtDepth = value,
           Depth = value)
Function = Erf(SymPos = value, Dose = value, ValueAtDepth = value,
           Depth = value)
```

By default, `SymmetryPosition=0`.

When `Function=Erf`, `Factor=0.8` in `LateralFunction` by default.

Some parameters have abbreviations (in parentheses) you can use, such as:

- `MaxValue` (`MaxVal`)
- `SymmetryPosition` (`SymPos`)

Specifying a 1D External Profile

To specify a 1D external profile, use the following primary parameters:

```
Function = subMesh1D(Datafile = "string", DataScale = Length = value),
                     Scale = Length = value), Range = line [(x1), (x2)])
```

where:

- `Datafile` specifies a file in XGRAPH format, which consists of a title enclosed in double quotation marks and a list of `"x y"` values. More than one profile can be included in `Datafile`.
- `DataScale` scales the data values contained in the data file. Each input value is multiplied by the `DataScale` factor. By default, `DataScale=1`.
- `Scale` scales the coordinate values from the file. By default, `Scale=1`.
- `Range` is optional and selects a range of values from the file. The keywords `x1` and `x2` must be given in the file coordinate system. `Range` is applied to all profiles in the file. By default, the entire data range is selected.

When `Function=subMesh1D`, `StandardDeviation=0.8` in `LateralFunction` by default.

Using the General Function Evaluator

The general function evaluator can be used in either of the following ways:

- Using `Eval`: A user-specified analytic function in the primary direction (normal to the reference window) and a separate decay function (Gaussian, error function, or a user-defined function with `Eval`) in the lateral direction. The syntax is:

```
AnalyticalProfile "reference name" {
   Function = Eval(init = "string", function = "string",
                   value = value)
   LateralFunction = Eval(init = "string", function = "string")
}
```

- Using `General`: A user-defined function specified directly in device coordinates. There is no concept of primary and lateral directions because the `General` function is specified directly as a function of the x-, y-, and z-directions. The syntax is:

```
AnalyticalProfile "reference name" {
   Function = General(init = "string", function = "string",
                      value = value)
}
```

Note:
`General` does not require `LateralFunction` since the `General` function is evaluated directly in all device coordinates.

Both the `Eval` and `General` functions use the same syntax for the primary parameters. The difference is that `General` uses spatial coordinates and `Eval` uses coordinates that are measured in the primary or lateral profile direction when used to define the primary or lateral profile, respectively.

The parameters `init`, `function`, and `value` correspond to the initialization formula, the evaluation formula, and the default value (in the case of a failed formula evaluation at a data point) (for the use of `General` functions, see Using Analytic Functions for Refinement I on page 123):

`init`

Specifies a semicolon-separated list of assignments for variables that are used later. This string is evaluated only once. For example:

```
init = "a=2;b=4"
```

`function`

Specifies an expression that is evaluated for every query. The variable that replaces the primary or lateral distance must be called `x`.

For example:

```
function = "sin(x)"
function = "exp(4*x)*sin(x)"
```

In general, 1D, 2D, and 3D simple analytic functions can be specified here. The variables `x`, `y`, and `z` can be used to refer to the respective x-, y-, and z-spatial coordinates.

`value`

Specifies the default return value if the evaluation fails. The default is `1.0e18` for the primary direction and `1` when used as `LateralFunction`.

Note that:

- All defined variables are global variables. This means that, if `init="a=1"` is defined in one function, the same value is used in all functions. Resetting the variable value in another function command has no effect.
- You can freely mix `Eval` with `Gauss`, `Erf`, and `subMesh1D` functions.
- The symbols `"pi"` and `"e"` can be used in the expressions.
- The following functions can be used:

  ```
  "sin", "cos", "tan", "asin", "acos", "atan", "sinh", "cosh", "tanh",
  "exp", "log", "log10", "sqrt", "floor", "ceil", "abs", "hypot", "deg",
  "rad"
  ```

- Numeric exponential constants can be specified as either `"2*10^18"` or `"2e18"`.
- As an extension to the `Eval` function, the `General` function assesses device coordinates directly, (x, y) and (x, y, z), and does not use primary and lateral distances. Any lateral functions and reference geometries (in the `Placements` section) are ignored.

Defining Constant Profiles

To define a constant profile, use the following syntax:

```
Definitions {
   Constant "reference name" {
      Species = "string"
      Value = value
   }
}
```

where:

- `Species` specifies the species or variables for the constant profile.
- `Value` specifies the value of the constant profile.

Defining Multibox Regions

Note:

Using the `Multibox` subsection is not recommended. Instead, use interface refinement with `MaxLengthInterface` (see Defining Refinement Regions on page 26).

A multibox is a special refinement box that specifies a graded refinement along the x-, y-, or z-direction. You can specify the required minimum and maximum element sizes, and an

additional refinement ratio in all directions. The created mesh is graded by using the specified ratios (also observing the minimum and maximum element sizes).

To define a multibox refinement region, use the following syntax:

```
Definitions {
   Multibox "reference name" {
      MaxElementSize = value | vector
      MinElementSize = value | vector
      Ratio = (ratio_width, ratio_height, ratio_depth)
   }
}
```

Parameters

`MaxElementSize, MinElementSize`

See description in Defining Refinement Regions on page 26.

`Ratio`

Controls the grading of the element sizes:

- *`ratio_width`* is the grading factor in the x-direction.
- *`ratio_height`* is the grading factor in the y-direction.
- *`ratio_depth`* is the grading factor in the z-direction (3D only).

Defining Particle Profiles

Particle definitions can be used to define profiles associated with discrete dopant distributions obtained from kinetic Monte Carlo (KMC) simulations using Sentaurus Process Kinetic Monte Carlo (Sentaurus Process KMC). A continuous profile is obtained from the discrete dopant distribution by associating a doping function with each discrete dopant (see Appendix B on page 162).

To define a particle profile, use the following syntax:

```
Definitions {
   Particle "reference name" {
      AutoScreeningFactor
      BoundaryExtension = float
      Divisions = value
      DopingAssignment = "CIC" | "NGP" | "Sano"
      Normalization
      NumberOfThreads = integer
      ParticleFile = "string"
      ScreeningFactor = value
      ScreeningScalingFactor = integer
      Species = "string"
```

```
    }
}
```

Parameters

Default values are given in parentheses if applicable.

`AutoScreeningFactor`

If this option is specified, Sentaurus Mesh calculates automatically a screening factor for each discrete dopant based on the local density of dopants using $k_c = 2N(x_0, y_0, z_0)^{1/3}$, where $N(x_0, y_0, z_0)$ is the density at the location of the discrete dopant.

Even when this option is specified, `ScreeningFactor` also must be specified because, when calculating the local density, the integration box size (in micrometers) is determined using (4.4934/`ScreeningFactor`) × 10^4.

`BoundaryExtension`

This parameter applies to 2D structures only and is used to obtain continuous doping on a 2D structure from a 3D KMC TDR file containing particle information.

The value given in micrometers is a thickness that is used internally to create an imaginary 3D structure by extruding the input 2D structure. This 3D structure is used to compute doping information, and this information is transferred to the 2D mesh.

`Divisions (10)`

This parameter applies to 2D structures only and is used in conjunction with the `BoundaryExtension` parameter. For each mesh point in a 2D structure, a number of points equal to a number of divisions, each separated by an equal amount, is created in the z-direction. The amount of separation is obtained by dividing the boundary extension with the number of divisions. The doping is computed on all of these points, and an average doping is assigned for the corresponding 2D mesh point.

`DopingAssignment ("Sano")`

The basic refinement method is the Sano method, but this parameter allows you to choose a method by which doping is assigned to a mesh immediately before saving the mesh:

- The cloud-in-cell (`"CIC"`) method distributes the doping of a particle to the vertex nodes of the element in which the particle is located.
- The nearest grid point (`"NGP"`) method assigns the doping of a particle to the nearest mesh node.
- The `"Sano"` method uses a doping function described in Appendix B on page 162 to distribute the doping of a particle to surrounding nodes.

`Normalization`

Specifying this option compensates for doping loss of dopants located near the boundary.

`NumberOfThreads (1)`

Parallelizes the local screening factor computation. Multithreading is recommended if the simulation contains thousands of particles. You can use the `--max_threads` command-line option to override the value specified by the `NumberOfThreads` parameter.

`ParticleFile`

Specifies the name of the KMC TDR file that contains the particle (discrete dopant) information.

`ScreeningFactor`

This is the cut-off parameter, k_c, for the doping function associated with each discrete dopant (see Appendix B on page 162). The `ScreeningFactor` (given in units of cm^{-1}) can be used as a fitting parameter; however, a value for it can be estimated from $k_c = 2N^{1/3}$, where N is the impurity concentration.

`ScreeningScalingFactor`

Controls the degree of smoothness of the profile. It is applied to the screening factor when `AutoScreeningFactor` is specified.

`Species`

Specifies the name of an active impurity concentration to associate with this definition, for example, `ArsenicActiveConcentration` and `BoronActiveConcentration`.

If `Species` is not specified, all active impurities that are found in the KMC particle file will be associated with this definition.

Defining Refinement Regions

To define a refinement region, use the following syntax:

```
Definitions {
   Refinement "reference name" {
      MaxElementSize = float | vector
      MinElementSize = float | vector
      RefineFunction = MaxGradient(parameters) |
                       MaxInterval(parameters) |
                       MaxLengthInterface(parameters) |
                       MaxLenPointCloud(parameters) |
                       MaxTransDifference(parameters)
   }
}
```

Parameters

Default values are given in parentheses if applicable.

`MaxElementSize (1)`

Controls the maximum size of grid elements (you also can use its abbreviation `MaxElemSize`). A real number or a vector $\vec{x} = [x_1, \ldots, x_d]$ can be specified, where d is the dimension and x_d represents the maximum edge lengths along the coordinate axes. A vector can be used to refine nonisotropically. Only values greater than zero are considered.

`MinElementSize (0.02)`

Controls the minimum size of grid elements (you also can use its abbreviation `MinElemSize`). A real number or a vector $\vec{x} = [x_1, \ldots, x_d]$ can be specified, where x_d represents the minimum edge lengths along the coordinate axes. Grid elements can be refined in one direction if their edge length in that direction is greater than the specified value. Only values greater than zero are considered.

`RefineFunction (MaxTransDifference)`

Different functions are available:

- The `MaxGradient` (or use its abbreviation `MaxGrad`) function evaluates the gradient of a profile (`Variable`) in the element. The syntax is:

  ```
  RefineFunction = MaxGrad(Variable = "Dataset name",
                           Value = float | vector | tensor)
  ```

  If the gradient is greater than `Value` and the edge lengths are large enough, the element is refined.

- The `MaxInterval` function analyzes each edge in a refinement tree cell and refines the edge if the data values at the endpoints overlap a given interval and the edge is longer than the maximum edge length defined on the interval. The syntax is:

  ```
  RefineFunction = MaxInterval(Variable = "Dataset name",
                       cmin = float | vector | tensor,
                       cmax = float | vector | tensor,
                       targetLength = float,
                       scaling = integer, rolloff)
  ```

  If the values at the edge endpoints overlap the range given by `cmin` and `cmax`, the algorithm checks only whether the edge length is shorter than the `targetLength` value. If this happens, the edge is split.

  When the edge is outside the value range and `rolloff` is specified, the tool adjusts `targetLength` to have a smooth transition into the coarser areas. To do this, the tool applies the following formula:

  ```
  targetLengthOutside = targetLength*(1 + log(Ca) – log(Cb))^2 *
                           scaling
  ```

where `Ca` and `Cb` are the variable values at the endpoints of the edge.

- The `MaxLengthInterface` (or use its abbreviation `MaxLenInt`) function produces refinement at interfaces. The syntax is:

```
RefineFunction = MaxLenInt(Interface("Material1","Material2"),
                    Value = value, Factor = value, Brep="string"
                    DoubleSide, UseRegionNames)
```

`RefineFunction` can be repeated for different interfaces in the same `Refinement` subsection.

The material specified in the `Interface` statement must be a valid DATEX material. The first material indicates the side of the interface on which the refinement is performed. To apply refinement to both sides of the interface, specify the `DoubleSide` option.

By default, interfaces are defined by a pair of materials. However, if the option `UseRegionNames` is used, the interface is interpreted as a regionwise specification.

The material `"All"` can be used to specify all interfaces of a given material and an empty string can be used to specify outer interfaces. In addition, the second argument in an interface specification can be a contact indicated by either the string `"Contact"` or the name of the contact (if `UseRegionNames` is specified).

The parameter `Brep` specifies a template boundary to be used to generate interface refinement on the current boundary implicitly, using the interfaces defined in the template boundary. This is also referred to as the *grain brep*.

Note:

The `MaxLenInt` function with the `Brep` parameter can only accept materials as interfaces. Each grain in the grain brep has a distinct region.

When you specify `Brep`, the `Interface` parameter has a different meaning: `Interface("Material1","Material2")`. To generate the interface refinement implicitly, you must follow these steps:

1. Specify the materials in which you want the refinement to be generated as the first entry to the `Interface` parameter (`"Material1"`). This is the *target boundary*.
2. Specify the materials of the grains as the second entry to the `Interface` parameter (`"Material2"`). This is the *template boundary*.

For example, to generate an implicit refinement on all the materials encompassed by the template boundary, specify:

```
Interface("All", "GrainMaterial")
```

If `Interface` is not defined, then no interface is refined. If `Value` is not specified, then it defaults to 1. The `Factor` parameter must be a number greater than or equal to 1.

If `Factor` is not defined, then it defaults to a huge number, so only one layer is produced.

Note:

The refinement that `MaxLenInt` performs is calculated by using the distance from an axis-aligned edge to the nearest interface specified in this subsection. For this reason, small features in the boundary might be missed by the refinement algorithm.

For example, in Figure 1 on page 30, the algorithm misses a portion of the interface refinement since it determines that the nearest interface is horizontal and not the small vertical step. Therefore, it considers the refinement conditions are satisfied.

- The `MaxLenPointCloud` function produces refinement at points belonging to a point cloud. The refinement is such that the point cloud is denser near the point locations and becomes coarser as you move away from those points. The syntax is as follows:

```
RefineFunction = MaxLenPointCloud(filename= "string",
                                  Value= value, Factor= value)
```

The parameter `filename` specifies a text file that consists of a list of 3D points, which would resemble trap sites. You also can generate a point cloud automatically by using the `ParticleGenerator` subsection (see Generating Particles on Interfaces on page 70). Alternatively, you can specify this file with the precise locations of trap sites.

The parameter `Value` is used to refine the edge if its length is greater than the specified value. If you do not specify this parameter, then it defaults to 1.

The parameter `Factor` specifies the growth factor for layers that grow away from the points. If you do not specify this parameter, then it defaults to a huge number so that only one layer is generated.

- The `MaxTransDifference` (or use its abbreviation `MaxTransDiff`) function evaluates the maximum difference of the transformed values of a profile at the vertices of the element. If the difference is greater than `Value` and the edge lengths are large enough, the element is refined. The syntax is:

```
RefineFunction = MaxTransDiff(Variable = "Dataset name",
                              Value = float | vector | tensor)
```

The transformation applied to the values used in the refinement functions (linear, logarithmic, arsinh) is defined in the `datexcodes.txt` file for each `Variable` (see *Utilities User Guide*, Variables).

`RefineFunction` can be repeated for different variables in the same `Refinement` subsection. If `Variable` is not defined, the default is `"DopingConcentration"`. If `Value` is not specified, it defaults to 1; however, no `RefineFunction` is assigned by default.

`Variable` defines the dataset used to adapt the grid. The grid can be adapted according to species or any type of variable defined in the output file. Values are computed from the analytic formulas, constant data, and external simulation results defined in the command file. Therefore, the name of a variable must match the name of a variable stored in the output file. The variable name must be enclosed in double quotation marks.

The parameter `Value` can be used to refine scalar, vector, or tensor variables.

To refine on a vector variable, use a vector of values, one per direction. For example, in two dimensions, you can refine on `ElectricField` as follows:

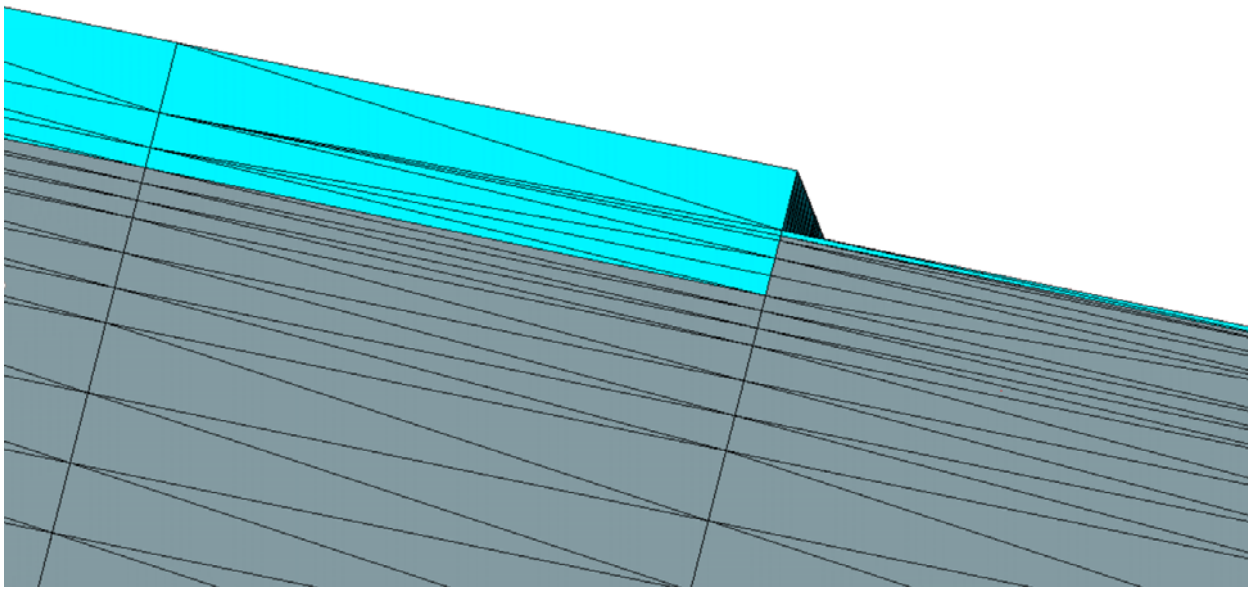
```
RefineFunction = MaxTransDiff(Variable = "ElectricField",
                              Value = (100,100) )
```

To refine on a tensor variable, use an array of nine elements where each component is represented like this: `(xx xy xz yx yy yz zx zy zz)`. Alternatively, if the tensor field has symmetric components, use a six-element array such as `{xx xy yy yz zx zz}`. For example, you can refine on `Stress` as follows:

```
RefineFunction = MaxTransDiff(Variable = "Stress",
            Value = (1e10 2e10 1e10 1e8 1e10 1e10 2e10 1e10 1e8))
```

The refinement is applied independently to each component of the vectors and tensors.

Figure 1 Missing vertical interface refinement around small step in the geometry



Defining Submeshes

External simulation results given on a mesh can be used to define profiles in the device. The external mesh must have the same spatial dimension as the device. The datasets defined on the external mesh are interpolated to the newly generated mesh. The external profiles are called *submeshes*. To define a submesh, use the following syntax:

```
Definitions {
   SubMesh "reference name" {
      Geofile = "string"
      ...
      Fields = ("string" [, float]), ("string" [, float]), ...
   }
}
```

Parameters

`Geofile`

Specifies the name of a file with an external mesh. The file must be in TDR format. The dimension of the external mesh must be the same as the dimension of the device.

Note:
Sentaurus Mesh uses a simplified version of the submesh syntax where only the `Geofile` parameter must be specified.

`Fields`

Specifies a list of fields to be extracted from the submesh. The other fields are ignored and are not used in the calculations or written to the output file. You can also specify an optional scaling factor (*`float`*) that scales the magnitude of the specified field *`float`* times. By default, all of the specified fields have a scaling factor value of 1.0.

Placements Section

The `Placements` section consists of sets of refinement and profile instances. Their positions in the device must be specified, and they must reference a definition given in the `Definitions` section. In other words, each instance or subsection consists of the instance name, an opening brace, the specification of parameters, and a closing brace.

The order of refinement regions in this section is important. The mesh generators select which refinement condition will be applied depending on the order of refinement regions described in the `Placements` section.

Syntax of Placements Section

```
Placements {
   AnalyticalProfile "instance name" {parameters}
   Constant "instance name" {parameters}
   Multibox "instance name" {parameters}
   Particle "instance name" {parameters}
   Refinement "instance name" {parameters}
   SubMesh "instance name" {parameters}
   ...
}
```

Note:

The order of profile instances in the `Placements` section is important only when the `Replace` option is used.

Geometric Elements

To specify `Placements` sections, you must use geometric elements. These elements are geometric objects used to select or locate data, and they are not part of the grid elements. The coordinates of these objects are defined relative to the coordinates of the device.

The allowed geometric elements and the number of coordinate values that must be specified depend on the dimension n of the device.

Let $\vec{x} = [x_1, \ldots, x_d]$ denote a point. The following geometric elements are defined:

Point ($\vec{x}_1$)

Line ($\vec{x}_1, \vec{x}_2$)

Rectangle ($\vec{x}_1, \vec{x}_2$)

Polygon ($\vec{x}_1, \ldots, \vec{x}_m$), $m > 2$

Complex polygon ($lump_1(polygon_1(\vec{x}_1, ..., \vec{x}_m), ..., polygon_p(\vec{x}_1, ..., \vec{x}_m))$
$lump_1(polygon_1(\vec{x}_1, ..., \vec{x}_m), ..., polygon_p(\vec{x}_1, ..., \vec{x}_m))$), $m > 2$

Cuboid ($\vec{x}_1, \vec{x}_2$)

Polyhedron $\left\{ polygon_1(\vec{x}_1, \ldots, \vec{x}_m), \ldots, polygon_p(\vec{x}_1, \ldots, \vec{x}_m) \right\}$, $m > 2$

Simple polygons are closed internally by adding the line segment between $\vec{x}_1 = [x_1, \ldots, x_d]$ and $\vec{x}_m = [x_1, \ldots, x_d]$. Only simple closed polyhedra are allowed. All their faces must be described.

The `GeoBrep {string}` element accepts the path to a boundary representation (brep) in the form of a file. This geometric element is used to select or locate all the interfaces (excluding the exterior interfaces) present in the brep. When specifying analytic profiles using this

geometric element, you can observe the peak values at brep interfaces decay as you move away from the interfaces.

The `GeoPointCloud {string}` element is used to select a point cloud defined in the text file specified. The point cloud should be a 3D point cloud with x-, y-, and z-coordinates separated by commas. When specifying analytic profiles using this geometric element, you can observe the peak values, at point locations in the point cloud, decay as you move away from the points.

Complex polygons are composed of lumps. Each lump represents a separate subpolygon, possibly containing holes. The first polygon inside a lump is the outer contour of the lump, while the subsequent polygons represent holes inside the lump.

The following example demonstrates the use of the `complexPolygon` element and represents two separate loops, the first of which has a hole inside:

```
AnalyticalProfile "buried n-channel" {
   Reference = "buried n-channel"
   ReferenceElement {
      Element = complexPolygon [
         lump [polygon [( 1.0 0.0 2.0 ) ( 2.0 0.0 2.0 ) ( 2.0 1.0 2.0 )
                        ( 1.0 1.0 2.0 )]
               polygon [( 1.3 0.3 2.0 ) ( 1.6 0.3 2.0 ) ( 1.6 0.6 2.0 )
                        ( 1.3 0.6 2.0 )]
         ]
         lump [polygon [( 0.0 1.5 2.0 ) ( 0.5 1.5 2.0 ) ( 0.5 2.0 2.0 )
                        ( 0.0 2.0 2.0 )]
         ]
      ]
      Direction = negative
   }
}
```

Note:

All polygons defined in a `complexPolygon` element must be coplanar.

To describe a polyhedron with arbitrarily oriented faces, use polygons instead of rectangles.

Table 2 lists the geometric elements that you can use to specify different kinds of window in each dimension.

Table 2 Geometric elements for specifying windows

Function	1D	2D	3D
`EvaluateWindow` in `Placements` section for profiles	`line`	`rectangle, polygon`	`cuboid, polyhedron`
`ReferenceElement` in `Placements` section for analytic profiles	`point`	`line`	`rectangle, polygon`

Table 2 Geometric elements for specifying windows (Continued)

Function	1D	2D	3D
`RefineWindow` in `Placements` section for refinements	`line`	`rectangle, polygon`	`cuboid, polyhedron`

In addition to the above-defined geometric elements, Table 3 lists other non-geometric elements that can be used to specify windows in the command file.

Table 3 Non-geometric elements for specifying windows

Element	Syntax
Material element	`material [<list of DATEX material names>]`
Region element	`region [<list of region names>]`
Composite element	`element {<list of geometric elements>}`
Sweep element	`sweepElement {<sweep element parameters>}`

Refinement or evaluation windows can be restricted to work on a particular material or region using the keyword `material` or `region`. For `material`, the argument is a valid DATEX material name in brackets. For `region`, the argument is a valid (existing) region name in brackets (see Regionwise and Materialwise Refinement on page 100).

In addition, you can combine elements to build more complex elements called *composite elements*, which are useful when defining complex reference elements for analytic profiles (see Using Composite Elements on page 99).

Sweep elements can be used to create 3D profiles by sweeping 2D profiles in 3D space. There are two types of sweep element: path sweep and angle sweep (see Creating 3D Profiles From 2D Cross Sections on page 102).

Placing Analytic Profiles

The syntax for an analytic profile is:

```
Placements {
   AnalyticalProfile "instance name" {
      Reference = "string"
      ReferenceElement {
         Element = element
         Direction = positive | negative
      }
```

```
        EvaluateWindow {
           Element = geometric element | material [<list>] |
                     region [<list>]
           DecayLength = value | GaussDecayLength = value
        }
        LocalReplace
        NotEvalLine
        Replace
    }
}
```

Parameters

`Reference`

Specifies the analytic profile to use. Only references to analytic profiles are allowed.

`ReferenceElement`

The direction of the normal to the `ReferenceElement` defines the direction of the analytic profile (instead of `ReferenceElement`, you can use its abbreviation `RefElem`). When evaluating the function values, the mesh points of the newly generated mesh are projected to the `Element`. The distance in the normal direction is used to evaluate the `Function`. The distance of the projection to the boundary of `Element` is used to compute the `LateralFunction`. By default, values are computed on both sides of the `Element`. If `Direction` is specified, function values are computed only on the positive or negative side of the `Element`.

In 1D devices, `Element` is a point, and the `positive` and `negative` directions are given by the coordinate axis. In 2D devices, `Element` is a line and the positive direction is taken to the right of the line.

In 3D devices, `Element` can be either a rectangle or polygon. The normal for a rectangle must be one of the coordinate axes. The `positive` and `negative` directions are defined from this axis. A (planar) polygon can be arbitrarily oriented in three dimensions. The direction is defined by the order of the points defining the polygon. A polygon is considered correctly oriented if the side of the polygon, which is surrounded by points in a positive orientation, defines the positive direction. There is no default value for `Element` and `Direction`.

`EvaluateWindow`

Restricts the placement of the analytic profile to a particular window, material, or region. See description in Placing Constant Profiles on page 36.

`LocalReplace`

See description in Placing Constant Profiles.

`NotEvalLine`

If this option is specified, the profile is not evaluated at the location of the reference element. This can be useful for placing two identical analytic profiles back-to-back using opposite directions, but without evaluating the reference element twice.

`Replace`

See description in Placing Constant Profiles.

Placing Constant Profiles

The syntax for constant profiles is:

```
Placements {
   Constant "instance name" {
      Reference = "string"
      EvaluateWindow {
         Element = geometric element | material [<list>] |
                   region [<list>]
         DecayLength = value | GaussDecayLength = value
      }
      LocalReplace
      Replace
   }
}
```

Parameters

`Reference`

Specifies the reference constant to use. Only references to constant profiles are allowed.

`EvaluateWindow`

Defines the domain where the profile is evaluated and a decay length is applied in the vicinity of the window boundaries. You can specify the domain by using a geometric element (see Table 2 on page 33) or by referring to materials or regions. (Instead of `EvaluateWindow`, you can use its abbreviation `EvalWin`.)

The decay function reduces round-off errors. It can be either an error function or a Gaussian function. To use an error function, specify `DecayLength` (or you can use its abbreviation `DecayLen`). For a Gaussian decay function, specify `GaussDecayLength`.

If `EvaluateWindow` is not defined, the transition between profiles is abrupt. If `DecayLength=0`, no decay function is applied and the transition between `EvaluateWindow` and its vicinity is abrupt. If `DecayLength` is negative, the profile is not applied to points on the border of `Element`. By default, `DecayLength=0` for all profiles.

For analytic, constant, and particle profiles, the default value of `Element` is the bounding box of the device. For submeshes, the default value of `Element` is the bounding box of the submesh. See the equations in Appendix A on page 150 for details.

Note:

Avoid using `EvaluateWindow` when the profile is valid in the entire device and no decay function is required. The evaluation of a geometric element is time consuming.

The `DecayLength` and `GaussDecayLength` parameters do not apply to particle profiles.

`LocalReplace`

With the `Replace` option, all the computed species are set to zero and are set with the value corresponding to the given profile instance. With the `LocalReplace` option, only species defined in the corresponding `Definitions` section are set exclusively to zero and are recomputed using the current profile instance. Other species are not updated. Accordingly, the net doping contribution is updated. By default, `LocalReplace` is switched off.

`Replace`

In general, the values for each profile at each point of the newly generated mesh are computed as the sum of all profile instances defined in the `Placements` section. Instances are inspected in the same order as they are defined in the command file. If `Replace` is specified for a given instance, all current summed values are replaced by the value corresponding to the given profile instance. By default, `Replace` is switched off.

Placing Multibox Regions

In the `Placements` section, a multibox instance is specified by the keyword `Multibox`, followed by the name of the multibox window, an opening brace, the specification of parameters, and a closing brace. Several multibox instances can refer to the same set of multibox parameters.

The syntax for a multibox instance is:

```
Placements {
   Multibox "instance name" {
      Reference = "string"
      RefineWindow = geometric element
   }
}
```

Parameters

`Reference`

Specifies the reference to a previously defined multibox.

`RefineWindow`

Defines the location of the refinement instance in the device. Table 2 on page 33 lists the geometric elements that can be used. By default, `RefineWindow` is the bounding box of the device.

Note:
If `RefineWindow` is not specified, the refinement instance is used as the default region for the entire device.

Placing Particle Profiles

The syntax for placing particle profiles in the `Placements` section is:

```
Placements {
   Particle "instance name" {
      Reference = "string"
      EvaluateWindow {
         Element = material [<list>] | region [<list>]
      }
      LocalReplace
      Replace
   }
}
```

Parameters

`Reference`

Specifies the reference particle to use. Only references to particle profiles are allowed.

`EvaluateWindow`

See description in Placing Constant Profiles on page 36.

`LocalReplace`

See description in Placing Constant Profiles.

`Replace`

See description in Placing Constant Profiles.

Placing Refinement Regions

In the `Placements` section, a refinement instance is specified by a name, an opening brace, the specification of parameters, and a closing brace. Several refinement instances can refer to the same set of refinement parameters.

The syntax for a refinement instance is:

```
Placements {
   Refinement "instance name" {
      Reference = "string"
      RefineWindow = geometric element | material [<list>] |
                     region [<list>]
   }
}
```

Parameters

`Reference`

Specifies the reference to a previously defined refinement.

`RefineWindow`

Defines the location of the refinement instance in the device. (Instead of `RefineWindow`, you can use its abbreviation `RefineWin`.) By default, `RefineWindow` is the bounding box of the device. Table 2 on page 33 lists the geometric elements that can be used. In addition, you can specify regionwise or materialwise refinement, or both refinements (see Regionwise and Materialwise Refinement on page 100).

You can specify `RefineWindow` multiple times in a `Refinement` subsection. When more than one `RefineWindow` is present, Sentaurus Mesh refines only the common sections of the refinement windows. This can be used to restrict the refinement to the part of a refinement box lying inside a region or material. For example:

```
Refinement "Refinement along current flow under the oxide" {
   Reference = "Refinement along current flow only in Silicon"
   RefineWin = cuboid [ ( 4.4 0 1 ) , ( 7.6 1.8 3.5 ) ]
   RefineWin = material ["Silicon"]
}
```

Note:
If no `RefineWindow` is specified, the refinement instance is used as the default region for the entire device.

Placing Submeshes

The syntax for references to submeshes in the `Placements` section is:

```
Placements {
   SubMesh "instance name" {
      Reference = "string"
      EvaluateWindow {
         Element = geometric element | material [<list>] |
                   region [<list>]
         DecayLength = value | GaussDecayLength = value
      }
```

```
        Ignoremat
        LocalReplace
        MatchMaterialType
        Reflect = X | Y | Z
        Replace
        Rotation {
           Angle = value
           Axis = X | Y | Z
        }
        ShiftVector = vector
    }
 }
```

Parameters

`Reference`

Specifies the reference submesh to use. Only references to profiles that are defined as `SubMesh` are allowed.

`EvaluateWindow`

Restricts the placement of the submesh to a particular window, material, or region. See description in Placing Constant Profiles on page 36.

`Ignoremat`

If you specify this option, the material in submeshes is ignored. The standard behavior of submesh interpolation is that the interpolated value is only accepted if the point is in a region with the same material.

This option allows Sentaurus Mesh to always accept the interpolation. (By default, if the materials do not match, the closest region with the correct material is searched.) The default behavior is not checked. For example:

```
Placements {
   SubMesh "NoName_0" {
      Reference = "NoName_0"
      Ignoremat
   }
}
```

`LocalReplace`

See description in Placing Constant Profiles.

`MatchMaterialType`

When you specify this option, the submesh attempts to match equivalent material types (for example, semiconductor, insulator, conductor) instead of trying to match material names when looking up values from which to interpolate.

`Reflect`

Specifies a reflection perpendicular to the specified coordinate axis. The allowed axes depend on the dimension of the device. The reflection point (or line or plane) is placed at the specified coordinate axis.

`Replace`

See description in Placing Constant Profiles on page 36.

`Rotation`

Performs a counterclockwise rotation around the axis. The center of the rotation is the origin of the coordinate system. By default, `Angle=0`. By default, `Axis=Z` (for two and three dimensions). In one dimension, `Rotation` is not supported.

`ShiftVector`

Translates a submesh to a new location. The vector is specified as two or three coordinates enclosed by parentheses.

Note:
The `Reflect`, `Rotation`, and `ShiftVector` operations are performed in the order in which they occur in the command file. The final location and orientation of the submesh depends on this order.

Interpolate Section

This optional section controls data interpolation that is performed after the mesh generators have finished.

Syntax of Interpolate Section

```
Interpolate {
   interpolateElements = true | false
   interpolateInterfaces = true | false
   keepTotalConcentration = true | false
   lateralDiffusion = true | false
}
```

Parameters

Default values are given in parentheses if applicable.

`interpolateElements (false)`

Interpolates element-type (scalar and vector) datasets. The element-type datasets of the input submesh are interpolated on the generated grid. The default value ignores element-type datasets.

`interpolateInterfaces (false)`

Interpolates all datasets defined in the source submesh on the generated grid if the submesh has interface regions. The default value means there is no interface interpolation on the generated grid.

`keepTotalConcentration (false)`

Saves the `TotalConcentration` field in the output file. By default, Sentaurus Mesh does not save this field in the output file. Sentaurus Device can calculate this field, if necessary, based on the available dopants.

`lateralDiffusion (false)`

Allows lateral extension on analytic profiles like that performed by the Synopsys Taurus™ Medici tool. This parameter affects only profiles with rectangular reference elements and attenuates the lateral decay factor by taking into account the distance from the interpolated points to all sides of the rectangle (see Lateral Error Function on page 159).

AxisAligned Section

This section controls the axis-aligned mesh generator, which takes a brep of the device and a series of user-defined refinement criteria, and follows these steps to create a mesh:

1. It attempts to repair the boundary using a combination of decimation and reconstruction algorithms. The algorithms are controlled by the `Decimate` and `DelPSC` subsections of the `Boundary` section (see Boundary Section on page 63).
2. It produces an initial coarse discretization of the bounding box of the structure by applying the `xCuts`, `yCuts`, and `zCuts` parameters. This creates an initial tensor-like structure that is used as the basis for the user-defined refinement.
3. The basic mesh is refined using user-defined criteria described in Chapter 3 on page 96. Each box is bisected recursively until all resulting boxes meet the criteria specified by the user. During this process, the mesh generator ensures that criteria such as `maxAngle`, `maxAspectRatio`, and `maxNeighborRatio` are satisfied.
4. After the boxes have been refined, they are imprinted on the boundary, producing a surface axis-aligned pattern. At this stage, short surface edges and poor angles are eliminated using a combination of boundary decimation and boundary repair algorithms.
5. As the last step before delaunization, the boxes are merged with the boundary, ensuring that intersecting the boxes with the boundary does not produce an unbalanced mesh (that is, a short edge next to a long one). To control this, the algorithm uses the parameter `maxBoundaryCutRatio`.

Syntax of AxisAligned Section

```
AxisAligned {
   binaryTreeSplitBox = (floatlist)
   binaryTreeSplitFactorX = integer
   binaryTreeSplitFactorY = integer
   binaryTreeSplitFactorZ = integer
   convexTriangulation = true | false
   fitInterfaces = true | false
   hintBoxSize = float
   imprintAccuracy = float
   imprintCoplanarFacesOnly = true | false
   imprintCoplanarityAngle = float
   imprintCoplanarityDistance = float
   latticeCellSize = (float float float)
   latticeDimensions = (integer integer integer)
   maxAngle = float
   maxAspectRatio = float
   maxBoundaryCutRatio = float
   maxNeighborRatio = float
   overscan = true | false
   overscanResolution = float
   skipSameMaterialInterfaces = true | false
   smoothing = true | false
   spacingMethod = even | regular | smooth
   splitDisconnectedRegions = true | false
   virtualSpacing = true | false
   xCuts = (floatlist)
   yCuts = (floatlist)
   zCuts = (floatlist)
}
```

Parameters

Default values are given in parentheses if applicable.

`binaryTreeSplitBox`

Specifies a box that defines the region where `binaryTreeSplitFactorX`, `binaryTreeSplitFactorY`, and `binaryTreeSplitFactorZ` are applied. By default, no box is used.

For 2D simulations, `binaryTreeSplitBox` is set to:

```
(xmin <float> ymin <float> xmax <float> ymax <float>)
```

For 3D simulations, `binaryTreeSplitBox` is set to:

```
(xmin <float> ymin <float> zmin <float> xmax <float> ymax <float>
 zmax <float>)
```

`binaryTreeSplitFactorX (1)`

Instructs Sentaurus Mesh to split the final binary tree used in the refinement step by a specified factor in the x-direction. This factor must be a power of 2; otherwise, the nearest power of 2 will be used. This parameter can be used to achieve an approximately uniform mesh refinement in the x-direction.

`binaryTreeSplitFactorY (1)`

Same as `binaryTreeSplitFactorX` but in the y-direction.

`binaryTreeSplitFactorZ (1)`

Same as `binaryTreeSplitFactorX` but in the z-direction.

`convexTriangulation (false)`

Creates a minimum triangulation of a 3D model containing convex regions. The input 3D boundary must be a convex model. Since the goal is to create a minimum triangulation of the convex model, refinement specifications in the command file (if any) are ignored. If the input model contains nonconvex regions, the meshing terminates with a corresponding message.

`fitInterfaces (false)`

Instructs Sentaurus Mesh to calculate the `xCuts`, `yCuts`, and `zCuts` automatically by first refining along the axis-aligned interfaces.

`hintBoxSize (1.0)`

When overscanning analytic profiles, Sentaurus Mesh calculates a *hint box* containing the peak value. The size of this box is a number of standard deviations from the peak. The default is one standard deviation around the peak value.

`imprintAccuracy (1e-5)`

Sets the distance used to determine whether two points are too close during axis-aligned imprinting.

`imprintCoplanarFacesOnly (true)`

If this option is set to `true`, Sentaurus Mesh imprints the axis-aligned refinement only on faces that are away from curved regions of the boundary. This is useful to avoid overrefinement in curved areas.

`imprintCoplanarityAngle (179.9)`

Sets the angle used by the face-imprinting algorithm to determine whether two boundary faces are coplanar.

`imprintCoplanarityDistance (1e-5)`

Sets the distance used by the face-imprinting algorithm to determine whether two faces are coplanar.

`latticeCellSize`

Specifies a tensor-like tessellation of the structure with a spacing that is as close as possible to the defined cell size. If `virtualSpacing=true`, then the generated lines guide the refinement algorithm, but not all of these lines will necessarily be present in the final mesh.

`latticeDimensions`

Specifies a tensor-like tessellation of the structure with the defined number of lines along each direction. If `virtualSpacing=true`, then the generated lines guide the refinement algorithm, but not all of these lines will necessarily be present in the final mesh.

`maxAngle (90` in 2D, `165` in 3D`)`

Determines the maximum angle produced in the binary tree.

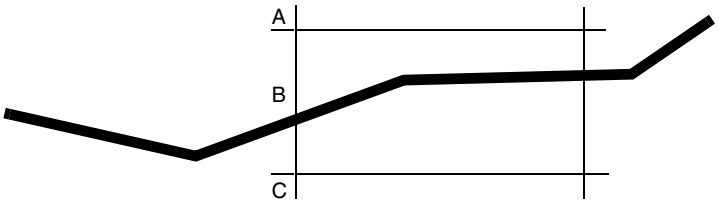
`maxAspectRatio (1e6)`

Specifies the maximum aspect ratio allowed in the elements of the binary tree at the end of the refinement step.

`maxBoundaryCutRatio (0.01)`

Defines the maximum-allowed ratio between adjacent segments on an axis-aligned box intersecting the boundary. When a segment belonging to an axis-aligned box intersects the boundary and the resulting cuts have a higher ratio than specified by this parameter, all axis-aligned faces associated with this segment are deactivated and are not allowed in the final mesh.

For example, in the following figure, a candidate axis-aligned segment AC can intersect a boundary edge at point B such that the AC segment becomes two collinear segments AB and BC.



In the absence of `maxBoundaryCutRatio` (value `0`), the ratio of the lengths of these edges would be unconstrained. With this parameter, the ratio between the short segment and the long segment are determined: either AB/AC or BC/AC. If either of these lengths is less than `maxBoundaryCutRatio`, the AC segment and any other segment connected to it are rejected and do not appear in the final mesh. In the case of a 3D mesh, this means that any axis-aligned face touching this segment is deactivated and cannot appear in the final mesh.

Setting `maxBoundaryCutRatio` to a high value (closer to 1) reduces the possibility of having sharp changes in mesh sizes across the boundary. However, at the same time, it

might create holes in the mesh since, potentially, many axis-aligned faces or segments can be deactivated. Setting `maxBoundaryCutRatio` to a small value reduces the possibility of holes around the boundary of the device, but it will produce sharp transitions in the mesh size at the boundary.

`maxNeighborRatio (2` in 2D, `4` in 3D`)`

Specifies the size ratio between adjacent elements.

`overscan (false)`

Instructs Sentaurus Mesh to scan the axis-aligned cells for field changes that justify more refinement based on the user parameters. The algorithm creates a small tensor mesh with the resolution indicated by the `overscanResolution` parameter and uses it to check for fine variations in the field profiles.

To avoid scanning all cells, the tool obtains ‘hints’ from the profiles as to where the interesting areas are located. For this purpose, the tool internally calculates the location of the peak values and p-n junctions, and gives them as a hint to the mesh generator.

`overscanResolution (0.3)`

Sets the resolution used to scan the device for field changes. The algorithm takes each unrefined cell and virtually subdivides it into smaller cells. Then, these small cells are checked for changes in the field values that justify more refinement.

`skipSameMaterialInterfaces (false)`

During refinement, if this parameter is set to `true`, Sentaurus Mesh ignores interfaces that have the same material on both sides.

`smoothing (true)`

Specifies whether the binary tree will be graded using the `maxAspectRatio` and `maxNeighborRatio` parameters.

`spacingMethod (even)`

Specifies the type of progression used by the refinement algorithm when expanding the refinement specified between lines:

- `even`: Distributes the cuts evenly, trying to approximate the spacing specified at the beginning of the interval.
- `regular`: Distributes the cuts evenly by using the exact spacing specified at the beginning of the interval, and leaving the last interval with an approximate size if there is no more room to accommodate the requested size.
- `smooth`: Distributes the cuts to have a smooth grading of spacing between lines.

`splitDisconnectedRegions (false)`

If an input boundary contains regions with multiple disconnected parts, this parameter specifies whether these regions should be split into multiple disconnected regions and

renamed according to Sentaurus Process naming rules, or whether these regions and their names should be preserved.

`virtualSpacing (false)`

Specifies whether the expansion lines produced by pairs of values defined by the `xCuts`, `yCuts`, and `zCuts` parameters will be either explicit lines or virtual lines that guide the refinement algorithm. When the lines are virtual, the refinement algorithm snaps the refinement coordinates to these lines instead of using the standard bisection algorithm. This allows the refinement to conform to a more user-defined pattern.

`xCuts, yCuts, zCuts`

These values represent refinement lines that are introduced into the mesh before any user-defined refinement. The lines define a rectilinear grid from which to start the refinement. Since each box in the initial grid is refined independently, different regions of the device can be isolated, thereby obtaining a more predictable refinement in each one of them. The cuts in each direction are specified as a list of cut points enclosed in parentheses.

Each cut point can be either a single floating-point value or a pair of floating-point values. A single value indicates the position of the cut point. When a pair of values is used, the first value indicates the position of the cut point, and the second value indicates the expansion of the cuts into a sequence of lines to be generated between adjacent pairs of lines.

The following example creates a series of grid lines located at 0, 1, and 2 μm. Between 0 and 1 μm, the series of lines should have a spacing of 0.1 (10 lines). Between 1 and 2 μm, the series of lines should have a spacing of 0.2 (five lines):

```
xCuts = ( (0 0.1) (1 0.2) 2)
spacingMethod = even
```

Note:

By default, no cuts are introduced into the mesh unless the `xCuts`, `yCuts`, or `zCuts` parameter is used.

Offsetting Section

The offsetting mesh generator uses the `Offsetting` section to create meshes with layers that follow the device interfaces. The layers are combined with the axis-aligned mesh generated by the axis-aligned mesh generator (see AxisAligned Section on page 42). The offsetting mesh generator first produces an axis-aligned mesh and then adds the offsetting layers on top of that mesh, clearing the axis-aligned elements that overlap the layers.

Note:

Specify either the `EnableOffset` or `EnableSections` option in the `IOControls` section of the command file to activate the offsetting mesh generator (see IOControls Section on page 16).

Syntax of Offsetting Section

```
Offsetting {
   # Offsetting global section:
   includeExterior = true | false
   noffset {
      factor = float
      hlocal = float
      maxlevel = integer
   }
   normalSmoothing = true | false

   # Offsetting interface section:
   noffset material | region "string" "string" {
      factor = float
      hlocal = float
      window = [(float float float) (float float float)]
   }

   # Offsetting region section:
   noffset material | region "string" {
      maxlevel = integer
   }
}
```

Parameters

Default values are given in parentheses if applicable.

`factor (1.3)`

As the front progresses, the thickness of the layers increases by this factor.

`hlocal (0)`

Sets the thickness of the first layer in µm. The default `hlocal=0` means no layering at all.

`includeExterior (false)`

This parameter is used to give a clearer definition for the keyword `"All"` when specifying offsetting regions or materials. When `includeExterior=true`, the keyword `"All"` specified in regions or materials will include the exterior region or material.

`maxlevel (200)`

Sets the number of layers that the offsetting mesh generator creates. If the front collides with other fronts or surfaces, the mesh generator stops prematurely.

`normalSmoothing (false)`

Reorients the surface normals around high curvature areas to avoid having tangled surfaces during the construction of analytic layers.

`window`

Specifies the cuboid used to confine the creation of layering. For a large interface, this parameter allows you to limit the layering to a spatial region of interest, thereby reducing the size of the grid. Note that `window` controls only the start of the layering and, therefore, layers might grow outside of the window. Multiple windows can be specified in an offsetting interface section.

The parameters `factor` and `hlocal` affect material interfaces and surfaces. They can be specified on an interface basis using the syntax with two region names. The syntax is not symmetric. For example:

- `noffset region "A" "B" {}` applies to the layers in region `A` where it borders region `B`.
- `noffset region "B" "A" {}` sets parameters for the other side of the same interface.

At places where contacts are defined, their names are used to define interfaces. The pseudo–region name `Exterior` is used for surfaces.

The parameter `maxlevel` can be set only per region using the syntax with one region name.

Note:

Consider the following:

- In both cases, you can use region names (keyword `region`) or the material property (keyword `material`). Since the material property is more persistent, it is better to use it instead of region names.
- Specify refinement criteria in the `Definitions` and `Placements` sections of the command file. Otherwise, the resulting mesh will be very coarse.
- Use a small number of layers to reduce spurious refinements near curved interfaces during delaunization of the mesh.

Delaunizer Section

This section controls the behavior of the delaunization algorithms in Sentaurus Mesh.

Syntax of Delaunizer Section

```
Delaunizer {
   coplanarityAngle = float
   coplanarityDistance = float
   delaunayTolerance = float
   edgeProximity = float
   faceProximity = float
```

```
    maxAngle = float
    maxConnectivity = float
    maxNeighborRatio = float
    maxPoints = integer
    maxSolidAngle = float
    maxTetQuality = float
    minAngle = float
    minDihedralAngleAllowed = float
    minEdgeLength = float
    minEdgeLengthAllowed = float
    sliverAngle = float
    sliverDistance = float
    sliverRemovalAlgorithm = integer
    storeDelaunayWeight = true | false
    type = boxmethod | conforming | constrained
}
```

Parameters

Default values are given in parentheses if applicable.

`coplanarityAngle (175)`

Determines whether two adjacent boundary faces are coplanar. The floating-point number represents the angle between the faces.

`coplanarityDistance (1e-5)`

Determines whether two adjacent boundary faces are coplanar. The floating-point number (given in µm) represents the absolute deformation made to the surface when the common edge is flipped.

`delaunayTolerance (1e-4)`

Specifies how close the ridges and boundary faces conform to the Delaunay criterion. A value of 0 everywhere implies a very strict Delaunay criterion. A value of 1 everywhere is equivalent to the construction of a constrained Delaunay triangulation (CDT). See Delaunay Tolerance on page 52.

`edgeProximity (0.05)`

Specifies the minimum ratio of the length of a new edge to the length of the parent edge from which it was generated. If an edge AB will be refined at point C and one of the ratios AC/AB or CB/AB is smaller than `edgeProximity`, point C is moved to the center of AB. When this value approaches 0.5, the edges will be more isotropically refined and the final mesh can contain many more points.

`faceProximity (0.05)`

Specifies the minimum ratio of the area of a new face to the area of the parent face from which it was generated. If a face ABC will be refined at point D and one of the ratios AD/r, BD/r, or CD/r is smaller than `edgeProximity` (where r is the radius of the circumscribed sphere), point D is moved to the Voronoï center of ABC. When this value

approaches 0.5, the faces will be more isotropically refined and the final mesh can contain many more points.

`maxAngle (180)`

Specifies the maximum angle allowed in the elements of the mesh (2D only).

`maxConnectivity (1000)`

Specifies the number of edges that can be connected to a mesh point.

`maxNeighborRatio (1e+30)`

Specifies the maximum-allowed ratio between the circumscribed spheres of neighboring elements. Values close to `2` should give a better grading, but they might also increase the mesh size considerably.

`maxPoints (500000)`

Sets a limit on the maximum number of points that the delaunizer generates. The limit is observed after the ridges have been recovered.

`maxSolidAngle (360)`

Specifies the maximum solid angle allowed in the elements of the mesh (3D only).

`maxTetQuality (1e37)`

Specifies the maximum circumscribed sphere radius–to–shortest edge ratio allowed in the mesh (3D only).

`minAngle (360)`

Specifies the minimum angle allowed in the elements of the mesh (2D only).

`minDihedralAngleAllowed (0)`

Checks that the 3D input boundary has a minimum dihedral angle greater than the value of `minDihedralAngleAllowed`. Otherwise, Sentaurus Mesh terminates with an error message (for 3D structures only).

`minEdgeLength (1e-9)`

Sets a value (in μm) that generates a warning message when the surface edges become shorter than this value.

`minEdgeLengthAllowed (0)`

Checks that the 3D input boundary has a minimum edge length greater than the value of `minEdgeLengthAllowed`. Otherwise, Sentaurus Mesh terminates with an error message (for 3D structures only).

`sliverAngle (175)`

Controls the elimination of slivers. The sliver elimination algorithm removes all elements where the maximum dihedral angle exceeds this value (given in degrees). The algorithm

endeavors to achieve this goal but, in general, it might not be possible. In practice, the final meshes contain elements where the maximum dihedral angle is approximately 179°.

`sliverDistance (1e-2)`

Controls the amount of damage performed by the sliver elimination algorithm (see Eliminating Slivers on page 148). The value specifies the maximum weight used at a given node.

`sliverRemovalAlgorithm (2)`

Selects the sliver elimination algorithm:

- `1` selects the original algorithm.
- `2` selects the algorithm that reduces the number of non-Delaunay elements by assigning more appropriate weights to vertices (see Eliminating Slivers on page 148).

`storeDelaunayWeight (true)`

Stores the nodal weight from the sliver elimination algorithm in the output file as a field variable when set to `true`. By supplying the Delaunay weight to Sentaurus Device, the box method library will have better convergence. This field variable is called the Delaunay–Voronoï weight (`DelVorWeight`) with the unit of μm^2 in the TDR file.

`type (boxmethod)`

Specifies the type of Delaunay mesh that the delaunization algorithm constructs:

- The `boxmethod` option imposes very strict conditions on the boundaries. The smallest circumscribed sphere around the boundary faces and ridges must be free of points.
- With the `conforming` option, the conditions at the boundary are more relaxed. This means that there exists a circumscribed sphere around a boundary face, which is free of points. This is equivalent to the standard Delaunay condition.
- When the `constrained` option is specified, boundary faces are inserted into a Delaunay mesh of the input points using a CDT algorithm. This option produces the least refinement of all options, but it produces meshes that are not suitable for device simulation.

Delaunay Tolerance

You can adjust the tolerance used to calculate the Delaunay criterion locally at interfaces, based on `material`, `region`, or `window` information. The syntax is:

```
boundary material | region "string" "string" {
   delaunayTolerance = float
   window = [ (float, float, float) (float, float, float) ]
}
```

```
surface material | region "string" {
   delaunayTolerance = float
   window = [ (float, float, float) (float, float, float) ]
}
```

The `delaunayTolerance` parameter is mandatory in the `boundary` and `surface` subsections. The `window` parameter is optional. These subsections do not accept any other parameters, that is, you cannot restrict the values of parameters such as `maxPoints` and `minEdgeLength` in regions or materials individually.

Examples

```
Delaunizer {
   # relax the tolerance at the boundary between any two materials
   boundary {
      delaunayTolerance=1
   }

   # restrict the tolerance at the boundary between silicon and oxide
   boundary material "Oxide" "Silicon" {
      delaunayTolerance = 1e-4
   }
}
```

Tensor Section

This section can contain the following subsections and controls the tensor-product mesh generator:

```
Tensor {
   Mesh {parameters}
   EMW {parameters}
   Box {parameters}
}
```

Note:

To activate the `Tensor` section, specify `EnableTensor` in the `IOControls` section of the command file (see IOControls Section on page 16).

If you specify `EnableSections` in the `IOControls` section and the `Tensor` section is present in the command file, then the axis-aligned and offsetting mesh generators, and the `Tools` section are deactivated, even if the corresponding `AxisAligned` section, `Offsetting` section, and `Tools` section are present. This means that only the tensor-product mesh generator will be executed, ignoring all other modules.

Controlling Mesh Generation

Various parameters can be defined in the `Mesh` subsection to control mesh generation. The `Mesh` subsection has the following syntax:

```
Tensor {
   Mesh {
      axisAlignedFeatureAngle = float
      doping
      grading = {float float float}
      grading off
      maxBndCellSize = float
      minBndCellSize = float
      maxCellSize = float
      minCellSize = float
      minNumberOfCells = integer
      numPoints = integer
      numPointsX = integer
      numPointsY = integer
      numPointsZ = integer
      scale = {float float float}
      window "string" float float float float float float
      xCuts = (floatlist)
      yCuts = (floatlist)
      zCuts = (floatlist)
   }
}
```

Parameters

Default values are given in parentheses if applicable.

`axisAlignedFeatureAngle (0.5 degrees)`

Part of the process of generating the refinement involves refining the grid at the location of axis-aligned interfaces found on the boundary. If there are boundary faces that are not nearly axis-aligned, the refinement algorithm ignores them, leading to unexpected holes in the refinement.

The `axisAlignedFeatureAngle` parameter allows you to specify a tolerance to indicate to the tool which faces should be considered axis aligned. The tool measures the deviation between the face normal and the nearest coordinate axis. If the deviation is smaller than `axisAlignedFeatureAngle`, the face is considered a feature and one of its points will be added to the coordinates used when refining the mesh.

`doping`

For Sentaurus Device Electromagnetic Wave Solver (EMW) applications, doping is generally not required.

When you specify the `EnableEMW` option in the `IOControls` section of the command file, doping is switched off to avoid unnecessary doping operations that can take too much

CPU time. If doping is required, specify the `doping` option to trigger doping. By default, for typical applications, doping is switched on.

`grading (1.25)`

Specifies the grading in each direction. The default is 1.25 in each direction. This can be specified in the following way:

- In three dimensions: `grading = {`*`gradx grady gradz`*`}`
- In two dimensions: `grading = {`*`gradx grady`*`}`

Note:
If you specify the `grading` parameter in both the `Mesh` subsection and the `EMW` subsection, then the `EMW` subsection takes precedence.

`grading off`

This statement switches off grading refinement. By default, grading is switched on.

`maxBndCellSize`

Sets the maximum cell size (given in µm) perpendicular to each material interface on the boundary. For this parameter, a normal vector is computed for each material interface on the boundary, and a direction of the maximum projection is found. Cells are clustered next to the material interface in the direction of the maximum projection. The default cell size in each direction is 10% of the geometry model length in that direction.

In addition, you can restrict `maxBndCellSize` to an interface by specifying an interface option as follows:

```
maxBndCellSize interface material | region "string" "string" float
```

To address external boundaries, you can use the keyword `"Exterior"` as one of the materials or regions of an interface. For example:

```
maxBndCellSize interface material "Silicon"   "Exterior" float
maxBndCellSize interface region   "substrate" "Exterior" float
```

By default, if neither material `"Exterior"` nor region `"Exterior"` is specified, exterior interfaces are not affected.

`minBndCellSize (1e-4)`

Sets the minimum cell size (in µm) perpendicular to a material interface on the boundary. For this parameter, a normal vector is computed for each material interface on the boundary, and a direction of the maximum projection is found. The cell size next to the material interface in the direction of the maximum projection will be, at least, the value of `minBndCellSize`.

In addition, you can restrict `minBndCellSize` to an interface in the same way as for `maxBndCellSize`.

`maxCellSize`

Sets the maximum cell size allowed in a region. The default cell size in each direction is 10% of the geometry model size in that direction.

`minCellSize (1e-4)`

Sets the minimum cell size (in µm) allowed in a region.

`minNumberOfCells (0)`

Sets the minimum number of cells required in each region and in each direction. The actual number of cells is not necessarily the same as the value of `minNumberOfCells` due to other parameters such as `maxCellSize` (default 10% of the entire structure) and `minCellSize` (default `1e-4` µm). The refinement algorithm for `minNumberOfCells` is based on adaptive bisection, so cell sizes are not necessarily equidistant. If you want the cell sizes to be equidistant, use `maxCellSize`.

`numPoints`

Specifies the fixed number of points in all directions.

`numPointsX`

Specifies the fixed number of points in the x-direction.

`numPointsY`

Specifies the fixed number of points in the y-direction.

`numPointsZ`

Specifies the fixed number of points in the z-direction.

`scale (1)`

Specifies a mesh scaling factor that can convert the mesh into different units. It can be specified as follows:

- In three dimensions: `scale = {`*`sx sy sz`*`}`
- In two dimensions: `scale = {`*`sx sy`*`}`

`window`

Restricts the effects of the refinement parameters. The following syntax is used to define a window:

- In three dimensions: `window "windowname"` *`xmin xmax ymin ymax zmin zmax`*
- In two dimensions: `window "windowname"` *`xmin xmax ymin ymax`*

`xCuts, yCuts, zCuts`

The values represent cuts in the tensor mesh where the refinement starts. These cuts are kept as part of the final tensor mesh. The cuts in each direction are specified as a list

of double-precision values enclosed in parentheses. By default, no cuts are introduced into the tensor mesh.

You can specify these refinement parameters (except for `numPoints`, `numPointsX`, `numPointsY`, and `numPointsZ`) for a region, material, direction, or window, in any of the following ways:

For all regions, in all directions	`parameter = floatOrint`
In a region, in all directions	`parameter region "regionname" floatOrint`
In a material, in all directions	`parameter material "materialname" floatOrint`
In a window, in all directions	`parameter window "windowname" floatOrint`
For all regions, in a direction	`parameter direction "x | y | z" floatOrint`
In a region, in a direction	`parameter region direction "regionname" "x | y | z" floatOrint`
In a material, in a direction	`parameter material direction "materialname" "x | y | z" floatOrint`
In a window, in a direction	`parameter window direction "windowname" "x | y | z" floatOrint`

Sometimes, the same parameter is assigned a value multiple times, in which case, the last assignment is taken into consideration.

Before specifying a parameter to be applied to a window, that window must be defined inside a `Mesh` subsection of the command file. If a parameter specified through the `window` option overlaps parameters specified with other options, the smallest of these parameters is considered while meshing.

Computing Cell Size Automatically

This application applies to EMW. When generating tensor meshes, maximum cell sizes are computed automatically when you specify the `EnableEMW` option in the `IOControls` section of the command file. The size computed is a function of the wavelength, the nodes per wavelength, and the magnitude of the complex refractive index (CRI).

You specify the required parameters in the `EMW` subsection of the `Tensor` section of the command file.

The `EMW` subsection has the following syntax:

```
Tensor {
   EMW {
      AllowThinLayerRemoval
      CRI material "materialName" CRIMODEL "string"
      CRI material "materialName" WavelengthDep Real | Imag | Real Imag
      CRI region "regionName" CRIMODEL "string"
      CRI region "regionName" WavelengthDep Real | Imag | Real Imag
      CRI WavelengthDep Real | Imag | Real Imag
      CRIMIPATH = "string"
      CRIMODEL = "string"

      grading = {float float float}
      grading off

      maxnpw material "materialName" float
      maxnpw material direction "materialName" "x" | "y" | "z" float
      maxnpw region "regionName" float
      maxnpw region direction "regionName" "x" | "y" | "z" float
      maxnpw = float
      maxnpwx = float
      maxnpwy = float
      maxnpwz = float

      NoEMWSolverConstraintsCheck

      npw material "materialName" float
      npw material direction "materialName" "x" | "y" | "z" float
      npw region "regionName" float
      npw region direction "regionName" "x" | "y" | "z" float
      npw = float
      npwx = float
      npwy = float
      npwz = float

      parameter filename = "string"

      wavefrequency = float
      wavelength = float
   }
}
```

Parameters

Default values are given in parentheses if applicable.

`AllowThinLayerRemoval`

Very thin layers might actually require a smaller cell size than that specified in the `minCellSize` (or `maxnpw`) setting to be resolved properly in the tensor mesh.

Therefore, thin layers can be the source of introducing cells with a smaller size than actually requested by users.

If you specify the `AllowThinLayerRemoval` option, Sentaurus Mesh removes these very thin layers in favor of maintaining the `minCellSize` setting.

`CRI material "`*`materialName`*`" CRIMODEL "`*`string`*`"`

This statement sets a CRI model for a specified material.

`CRI material "`*`materialName`*`" WavelengthDep Real | Imag | Real Imag`

This statement sets the wavelength dependency for this material.

`CRI region "`*`regionName`*`" CRIMODEL "`*`string`*`"`

This statement sets a CRI model for a specified region.

`CRI region "`*`regionName`*`" WavelengthDep Real | Imag | Real Imag`

This statement sets the wavelength dependency for a specified region. For example:

- Set the wavelength dependency for this region only on the real part of the CRI:

  ```
  CRI region "Silicon_0" WavelengthDep Real
  ```

- Set the wavelength dependency for this region only on the imaginary part of the CRI:

  ```
  CRI region "Silicon_0" WavelengthDep Imag
  ```

- Set the wavelength dependency for this region on both the real and imaginary parts of the CRI:

  ```
  CRI region "Silicon_0" WavelengthDep
  ```

`CRI WavelengthDep Real | Imag | Real Imag`

This statement sets the wavelength dependency on the real part, or the imaginary part, or both parts of the CRI values. Specifying the real and imaginary statements is optional. For example:

- Set the wavelength dependency only on the real part of the CRI:

  ```
  CRI WavelengthDep Real
  ```

- Set the wavelength dependency only on the imaginary part of the CRI:

  ```
  CRI WavelengthDep Imag
  ```

- Set the wavelength dependency on both the real and imaginary parts of the CRI:

  ```
  CRI WavelengthDep
  ```

`CRIMIPATH`

(Optional) Specifies the location of the CRI model.

`CRIMODEL`

(Optional) Specifies the name of the CRI model.

`grading (1.25)`

Specifies the grading in each direction. The default is 1.25 in each direction. This can be specified as follows:

- In three dimensions: `grading = {gradx grady gradz}`
- In two dimensions: `grading = {gradx grady}`

Note:
If you specify the `grading` parameter in both the `Mesh` subsection and the `EMW` subsection, then the `EMW` subsection takes precedence.

`grading off`

This statement switches off grading refinement. By default, grading is switched on.

`maxnpw`

Defines the maximum number of nodes per wavelength according to a material or region, and in a direction. This parameter computes a minimum cell size (by replacing `npw` with `maxnpw` in Equation 2 on page 61). Therefore, it can control the largest stable time step for EMW.

Note:
Too small cell sizes can increase the simulation time for EMW.

The following example defines the maximum nodes per wavelength in silicon material in the y-direction:

```
maxnpw material direction "Silicon" "y" 45
```

The `maxnpw` parameter is used only if you specify its value. For certain domains in complex structures, the `maxnpw` and `npw` values might be contradictory. If `maxnpw` < `npw`, then `maxnpw` = `npw`. Because of the nature of the structured tensor mesh, the actual cell size for some regions might not satisfy the specified `npw` or `maxnpw`. In this case, a notification is issued with the expected and the actual values for `npw` and `maxnpw`.

`maxnpwx`

Sets the maximum nodes per wavelength similar to `maxnpw` but only in the x-direction for all materials. If `maxnpwx` < `npwx`, then `maxnpwx = npwx`.

`maxnpwy`

Sets the maximum nodes per wavelength similar to `maxnpw` but only in the y-direction for all materials. If `maxnpwy` < `npwy`, then `maxnpwy = npwy`.

`maxnpwz`

Sets the maximum nodes per wavelength similar to `maxnpw` but only in the z-direction for all materials. If `maxnpwz` < `npwz`, then `maxnpwz = npwz`.

`NoEMWSolverConstraintsCheck`

Tensor meshes generated for EMW must not have holes inside the structure. By default, Sentaurus Mesh exits with an error if a hole exists inside the structure. Specifying the `NoEMWSolverConstraintsCheck` option deactivates this check.

`npw (10)`

Defines the number of nodes per wavelength according to a material or region, and in a direction. If you do not use the direction, the specified value applies in all directions. The following example defines the nodes per wavelength in silicon material in the x-direction:

```
npw material direction "Silicon" "x" 20
```

The default value of the nodes per wavelength is 10 in all directions for each material.

For a given material, the cell size is computed using the formulas:

$$\lambda_{\mathrm{mat}} = \frac{\mathrm{wavelength}}{R_{\mathrm{mod}}} \tag{1}$$

$$\mathrm{cellsize}_{\mathrm{mat}} = \frac{\lambda_{\mathrm{mat}}}{\mathrm{npw}} \tag{2}$$

In Equation 1, the parameter R_{mod} is computed depending on the settings of the CRI model:

- If `CRI WavelengthDep Real` is used, then $R_{\mathrm{mod}} = |\mathrm{n}|$.
- If `CRI WavelengthDep Imag` is used, then $R_{\mathrm{mod}} = |\mathrm{k}|$.
- If `CRI WavelengthDep Real Imag` is used, then $R_{\mathrm{mod}} = \sqrt{\mathrm{n}^2 + \mathrm{k}^2}$.

Here, n is the real part and k is the imaginary part of the CRI.

`npwx`

Sets the nodes per wavelength similar to `npw` but only in the x-direction for all materials.

`npwy`

Sets the nodes per wavelength similar to `npw` but only in the y-direction for all materials.

`npwz`

Sets the nodes per wavelength similar to `npw` but only in the z-direction for all materials.

`parameter filename = "`*`string`*`"`

This statement sets the parameter `filename` that contains the CRI table of materials that are present in the input structure.

`wavefrequency`

Specifies the value of the wavelength frequency. The wavelength is computed using this value and the speed of light.

`wavelength (0.555)`

Specifies the wavelength in micrometers.

Plotting

You can add new regions to tensor meshes that can be used for plotting purposes in EMW applications. The new regions are specified by using the `Box` subsection, and multiple `Box` subsections can be specified in the `Tensor` section. The `Box` subsection has the following syntax:

```
Tensor {
   Box {
      boundingBox
      boundingBox region = "string"
      endPoint = {float float float}
      exact = "yes" | "no"
      material = "string"
      name = "string"
      startPoint = {float float float}
      tolerance = {float float float}
   }
}
```

Parameters

Default values are given in parentheses if applicable.

`boundingBox`

You can use this option instead of specifying `startPoint` and `endPoint`. It automatically sets the minimum and maximum of the structure bounding box as the `startPoint` and `endPoint`, respectively.

`boundingBox region = "`*`string`*`"`

You can use this statement instead of specifying `startPoint` and `endPoint`. It automatically sets the minimum and maximum of the region bounding box as the `startPoint` and `endPoint`, respectively.

`endPoint`

Specifies the highest point of the bounding box used for the plot (*`xmax ymax zmax`*).

`exact ("no")`

If `exact="yes"`, the resultant mesh should contain nodes whose coordinates match `startPoint` and `endPoint`. If `exact="no"`, the nodes that are closest to `startPoint` and `endPoint` are written in the tensor mesh.

`material ("none")`

If not specified, the name of the material defaults to `"none"`.

`name`

Specifies the name of the region.

`startPoint`

Specifies the lowest point of the bounding box used for the plot (*`xmin ymin zmin`*).

`tolerance`

The tolerance is used only if `exact="yes"`. The tolerance value indicates that the box must be aligned to any existing boundary or cell interface within this tolerance distance. This avoids unnecessary small cells locally. A value of model length in each direction multiplied by 1e-4 µm is used as the default. This can be specified as follows:

- In three dimensions: `tolerance = {`*`tx ty tz`*`}`
- In two dimensions: `tolerance = {`*`tx ty`*`}`

Boundary Section

The `Boundary` section controls the boundary algorithms and can contain different subsections and their corresponding parameters.

Syntax of Boundary Section

```
Boundary {
   Decimation {
      apply = true | false
      accuracy = float
      coplanarityAngle = float
      minDihedralAngle = float
      shortEdge = float
   }
   DelPSC {
      apply = true | false
      accuracy = float
      ridgeAngle = float
      ridgeSampling = float
      skipFeatureSize = float
   }
```

```
   DualContouring {
      apply = true | false
      decimation = true | false
      minAngle = float
      minDihedralAngle = float
      resolution = float
   }
   RegionMismatch {
      allow = true | false
      minVolume = float
   }
   Save {
      filename = "string"
   }
}
```

Decimating 3D Boundaries

The `Decimation` subsection specifies the decimation of the 3D boundary. It is switched on by default.

The decimation process removes nodes from the surface, thereby generating a simpler structure. A node is removed only if the deformation caused by removing the node is less than the value specified by the `accuracy` parameter.

Parameters

Default values are given in parentheses if applicable.

`apply (true)`

Activates or deactivates decimation of a 3D boundary.

`accuracy (1e-6)`

Restricts the changes to the boundary that are undertaken by the decimation algorithm. The decimation algorithm cannot modify the boundary more than the value of `accuracy` given in µm.

`coplanarityAngle (175)`

Specifies the angle used during boundary decimation to determine whether two faces are coplanar.

`minDihedralAngle (10)`

Determines the minimum dihedral angle (in degrees) on the surface produced by the decimation algorithm.

`shortEdge (1e-7)`

Specifies the minimum edge length (in µm) produced on the boundary before the decimation step.

Applying the DelPSC Algorithm to Boundary Surfaces

The `DelPSC` subsection instructs Sentaurus Mesh to apply the Delaunay refinement for piecewise smooth complex (DelPSC) algorithm to boundary surfaces and controls the quality of the discretization.

The `DelPSC` subsection is useful for curved surfaces. It is switched off by default and is activated when this subsection is specified or `apply=true`.

Note:

If you specify `IOControls{numThreads=`*`integer`*`}`, then the DelPSC algorithm uses multithreading. See [1][2] for a description of the algorithm.

Parameters

Default values are given in parentheses if applicable.

`apply (false)`

Activates or deactivates the DelPSC algorithm.

`accuracy (0.0001)`

Controls the deviation (in μm) between the new curved surface and the original curved surface in the DelPSC algorithm. The new curved surface can deviate from the original curved surface by, at most, the value of `accuracy`. New vertices lie exactly on the original surface, but new triangles cannot lie exactly on the original surface *unless* the original surface is flat. In general, the smaller the value of `accuracy`, the smoother the new surface becomes, and the more accurate the new surface represents the original surface.

In general, setting `accuracy` to 2% of the radius of curvature is appropriate. Larger values allow the DelPSC algorithm to run faster but generate coarser discretization on curved surfaces. Smaller values make the DelPSC algorithm run more slowly and generate finer discretization on curved surfaces.

`ridgeAngle (150)`

Angle used by the DelPSC algorithm to determine geometric features.

`ridgeSampling (0.01)`

Controls the size (in μm) of small triangles on curved surfaces in the DelPSC algorithm.

In general, setting `ridgeSampling` to 10% of the radius of curvature is appropriate. Larger values allow the DelPSC algorithm to run more quickly but generate bigger triangles next to geometric features and triple lines on curved surfaces. Smaller values make the DelPSC algorithm run more slowly and generate smaller triangles next to geometric features and triple lines on curved surfaces.

`skipFeatureSize (0.0001)`

Specifies the minimum threshold for ridge lengths. This is used to ignore extremely small features.

Reconstructing the Boundary Using the Dual-Contouring Algorithm

The `DualContouring` subsection instructs Sentaurus Mesh to use the dual-contouring algorithm to reconstruct the boundary. It is switched off by default and is activated if this subsection is specified or `apply=true`.

Dual contouring can be useful when the input boundary is low quality, that is, it contains triangles with small surface angles or small dihedral angles. Dual contouring is designed to reconstruct the boundary with high geometric accuracy. It can preserve sharp or thin features as much as possible. It can also collapse small edges or thin material layers (see *Parameters*), thereby facilitating a more robust volume meshing. See [3][4][5] for a description of the algorithm.

Parameters

Default values are given in parentheses if applicable.

`apply (false)`

Activates or deactivates the dual-contouring algorithm.

`decimation (false)`

Instructs Sentaurus Mesh to apply a postprocessing decimation step to reduce the number of generated vertices.

`minAngle (2)`

Defines the minimum-allowable surface angle (in degrees) on the output surface.

`minCutLength (1e-6)`

Defines the minimum length (in μm) between consecutive cuts on a primal edge (octree edge). Cuts with a distance less than this threshold are merged. This parameter facilitates the collapse of thin features on the output surface. The acceptable range of values is (0, `resolution`). The higher the value, the more aggressive the collapse. Values greater than 0.5×`resolution` might compromise the geometric accuracy of the output surface.

`minDihedralAngle (10)`

Defines the minimum-allowable dihedral angle (in degrees) on the output surface.

`minDualEdgeLengthFactor (0.05)`

Defines a factor to compute the minimum length of a dual edge (that is, an edge on the output surface) relative to the minimum length of a primal edge (`resolution`). This parameter is used to collapse the short edges on the output surface. The acceptable range of values is (0, 1). The higher the value, the more aggressive the collapse.

`minPrimalSubedgeLengthFactor (0.05)`

Defines a factor to compute the minimum length of a primal sub-edge (octree sub-edge) relative to the minimum length of a primal edge (that is, `resolution`). The acceptable range of values is (0, 1). The higher the value, the more aggressive the collapse. Values greater than 0.5 might compromise the geometric accuracy of the output surface.

`resolution (0.002)`

Defines the minimum spacing (in µm) of a primal mesh (that is, an octree grid) used for the reconstruction. The smaller the value, the more accurately the output surface represents the input surface. A very small value can result in an excessive number of triangles and a longer runtime.

`smoothing (false)`

Applies an algorithm to reduce the noise on the output surface.

`snapBoundaryToBox (0)`

Snaps the vertices on the exterior output surface to the sides of the bounding box (defined by the input surface). Snapping is applied only if gas material is present in the input surface. Valid values are:

- A value of 0 deactivates snapping.
- A value of 1 allows snapping to all sides of the box.
- A value of 2 allows snapping to all sides of the box excluding the +z-side.

Allowing for Region Mismatch

If you specify the `RegionMismatch` subsection, when Sentaurus Mesh checks whether the number of regions in the input boundary and the number of regions at the end of the meshing process are the same, if the numbers of regions differ, then Sentaurus Mesh ignores the discrepancy, and the meshing process continues.

This subsection is switched off by default and is activated if it is specified or `allow=true`.

If you do not specify the `RegionMismatch` subsection, when Sentaurus Mesh checks whether the number of regions in the input boundary and the number of regions at the end of the meshing process are the same, if the numbers of regions differ, then the meshing process stops.

Parameters

Default values are given in parentheses if applicable.

`allow (false)`

Activates or deactivates checking for region mismatch.

`minVolume (0)`

Specifies a region volume that Sentaurus Mesh uses when checking the volumes of all deleted regions:

- If the volume of a deleted region is *less than the value* specified by this parameter, then the meshing process continues and the number of deleted regions is reported.
- If the volume of a deleted region is *greater than the value* specified by this parameter, then the meshing process stops.

Saving Results

The `Save` subsection is used to save a boundary file. It is switched off by default and is switched on if specified.

Parameters

Default values are given in parentheses if applicable.

`filename`

Specifies the name of the file in which to save the boundary.

Tools Section

You use the `Tools` section to execute geometric operations on either a boundary file or a mesh file. The input mesh can be either a tetrahedral mesh or a hybrid (mixed-element) mesh.

If a hybrid mesh is used, you must first convert it to a tetrahedral mesh before applying the tool (see Converting a Tetrahedral Mesh to a Hybrid Mesh on page 82). The mesh is converted back to a hybrid mesh after all operations have been executed. If an operation such as a simple transformation is applied, the resulting mesh might differ slightly from the original mesh, despite no topological changes.

Note:

Operations are executed according to their order in the `Tools` section. The output of one operation becomes the input for the next operation.

To use the `Tools` section, specify `EnableTools` in the `IOControls` section of the command file (see IOControls Section on page 16).

If you specify `EnableSections` in the `IOControls` section and the `Tools` section is present in the command file, then the axis-aligned and offsetting mesh generators are deactivated, which means no mesh will be created and only the `Tools` section will be executed, even if the corresponding `AxisAligned` and `Offsetting` sections are present. However, if the `Tensor` section is also present, then it overrides the `Tools` section, which will be ignored during the execution of the tool.

Syntax of Tools Section

```
Tools {
   Subsection {parameters}
}
```

Appending the Input Structure

This subsection appends the input structure periodically at the specified position:

```
Tools {
   Append {
      axis = xmin | ymin | zmin | xmax | ymax | zmax
      map "stringA" = "stringB"
   }
}
```

Note:

This operation works only for 2D and 3D boundaries.

Creating Profiles

This subsection creates profiles in the input mesh with the description given in the command file:

```
Tools {
   CreateProfiles {
      SrcMesh = "string"
      CmdFile = "string"
   }
}
```

The mesh is not modified in this process as the refinement specifications in the command file are ignored. During profile creation, the existing profiles in the input mesh, which are again specified in the command file, will only be recreated. The rest of the profiles are untouched.

Generating Particles on Interfaces

This subsection generates a point cloud with a specified minimum spacing on all interface planes in a 3D brep (interface planes lying on the exterior boundary are excluded). This feature is useful for simulating trap sites in polycrystalline grain devices. These traps are usually generated along the boundaries of the grain.

Using a brep generated by the `Voronoi` subsection of the `Tools` section (where each grain is represented by a unique region), the particle generator first identifies the grain boundary interfaces. Next, the generator calculates the number of traps needed for an interface using the user-defined `trapDensity` parameter and the interface area. Then, the generator generates a 2D grid along the interface and generates a random trap site within each grid. To ensure that no two trap sites are generated too close to each other, the generator ensures that there is at least a user-defined `minSpacing` distance between any two trap sites.

The syntax is:

```
Tools {
   ParticleGenerator {
      brepIn = "string"
      minSpacing = float
      pointCloud = float
      trapDensity = float
   }
}
```

The following parameters control the generation of particles on interfaces:

- `brepIn` is the input file of the granularized brep with each grain belonging to a separate region. This file is usually generated by using Neper [6] or the `Voronoi` subsection of the `Tools` section (see Generating Voronoï Diagrams on page 87).
- `minSpacing` constrains the minimum distance (in µm) between any two trap sites to the specified value. The default is 0.
- `pointCloud` represents the output file in a `.txt` format. The trap sites generated by the `ParticleGenerator` subsection are stored in this file. The x-, y-, and z-coordinates for each point in the point cloud are separated by commas.
- `trapDensity` is the user-specified areal density of traps in units of cm^{-2}. The default is 0.

Example: Simple Cube Geometry

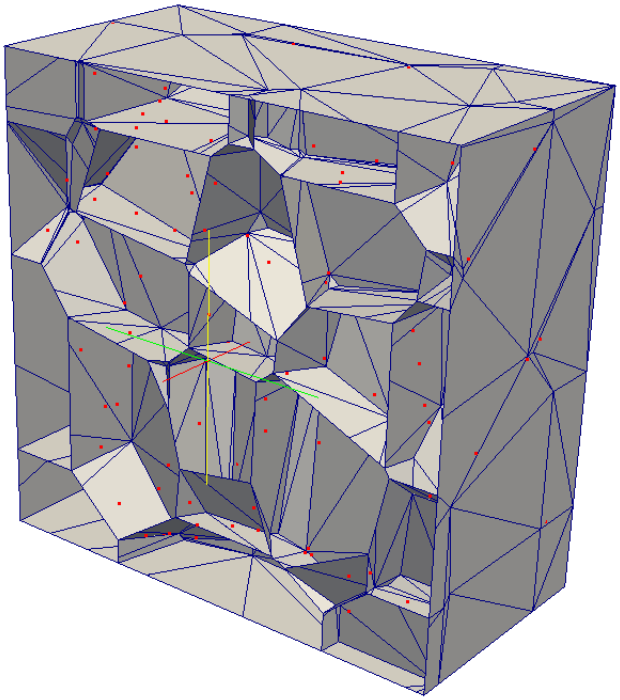
The following example demonstrates the `Tools` section of the command file for a simple cube geometry:

```
Tools {
   ParticleGenerator {
      brepIn = "output.tdr"
```

```
        pointCloud = "traps.txt"
        trapDensity = 2e+9
        minSpacing = 0.005
    }
}
```

The particle generator generates a point cloud that can be used as trap locations in a nonvolatile memory device simulation (see Figure 2).

Figure 2 *Simple cube (output.tdr) with random particles generated at interfaces (shown by red dots)*



These points are generated along the interfaces of the input granular structure, which can be generated by using Neper or the `Voronoi` subsection of the `Tools` section. The number of traps for an interface is determined by the `trapDensity` parameter (in cm^{-2}). The output is stored in a `.txt` file specified by the `pointCloud` parameter (see Figure 3 on page 72). You can further constrain the trap locations to have a minimum spacing between them by using the `minSpacing` parameter.

Figure 3 The traps.txt file with trap locations

```
1.238463,   1.843835,   1.213903
1.615727,   1.996153,   1.272495
1.891272,   1.655340,   0.827042
1.864380,   1.684536,   1.005161
1.984266,   0.916928,   1.982204
1.912939,   1.115834,   1.746156
1.751364,   0.973681,   1.784809
1.852335,   0.875770,   1.456516
1.969054,   0.924266,   1.277433
1.813912,   1.288538,   1.454003
1.797990,   1.259464,   1.295801
1.766881,   0.939762,   1.537297
1.766881,   1.093809,   1.530846
1.766881,   1.153133,   1.689808
1.766881,   0.909298,   1.731895
1.670283,   1.787621,   0.990017
1.702442,   1.787621,   0.817290
1.478407,   1.787621,   1.242116
1.483824,   1.787621,   0.868747
1.677305,   0.929452,   1.797036
1.718198,   1.320349,   1.166242
1.643375,   0.978992,   1.261466
1.093687,   0.156429,   1.576199
1.962947,   0.863177,   1.186962
0.329835,   1.002362,   0.312956
0.477516,   1.034192,   0.174974
```

Setting a Transformation

This subsection sets a transformation matrix to a mesh or a boundary:

```
Tools {
   Set Transformation {
      translation = (float float [float])
      scale = float | (float float float)

      rotation {
         axis = (float float float)
         angle = float
      } |
      rotation {
         matrix (float float float float float float float float float)
      }
   }
   Apply Transformation
}
```

Note:

The `Set Transformation` operation applies to both the mesh and the boundary.

A translation vector sets the translation. The rotation can be set by either an axis vector and an angle in degrees, or a matrix. A mesh or a boundary also can be scaled by specifying a floating-point value or a vector.

The `Apply Transformation` statement applies the specified transformation.

Removing Short Features

This subsection removes unwanted short features in a boundary:

```
Tools {
   Decimate {
      accuracy = float
      shortedge = float
   }
}
```

The `accuracy` parameter indicates the deviation of a structure from its original location. The default is `1e-8`.

The `shortedge` parameter removes short edges. All edges that are shorter than this parameter are eliminated. This parameter does not have a default value and is activated only when it has a nonzero value.

Note:

Short edges can also be removed by a boundary rediscretization with a dual-contouring algorithm (see Rediscretizing the Boundary Using the Dual-Contouring Algorithm on page 74).

Rediscretizing the Boundary Using the DelPSC Algorithm

This subsection rediscretizes the boundary by surface remeshing using the DelPSC algorithm that creates good-quality triangles on non-flat surfaces of the boundary:

```
Tools {
   DelaunaySurfaceRemeshing {
      DelPSCAccuracy = float
      DelPSCRidgeAngle = float
      DelPSCRidgeSampling = float
   }
}
```

The following parameters control the quality of the discretization:

- `DelPSCAccuracy` controls the deviation between the new curved surfaces and the original curved surfaces. The new curved surface can deviate from the original curved surface by, at most, the value of `DelPSCAccuracy`. New vertices lie exactly on the original surface, but new triangles cannot lie exactly on the original surface *unless* the original surface is flat. In general, the smaller the value of `DelPSCAccuracy` is, the smoother the new surface becomes, and the more accurate the new surface represents the original surface.
- `DelPSCRidgeAngle` is an angle computed at each edge. It is a dihedral angle between two shared faces. This parameter identifies the geometric features. Default: 95°
- `DelPSCRidgeSampling` discretizes the ridges and controls the size of small triangles. Default: 0.01 µm

Note:

The DelPSC algorithm uses multithreading if you specify `IOControls{numThreads=`*`integer`*`}` (see IOControls Section on page 16).

Rediscretizing the Boundary Using the Dual-Contouring Algorithm

This subsection rediscretizes the boundary by surface remeshing using the dual-contouring algorithm. Dual contouring can be useful when the input boundary is low quality, that is, it contains triangles with small surface angles or small dihedral angles. Dual contouring is designed to reconstruct the boundary with high geometric accuracy and to preserve sharp or thin features. It can also be used to collapse small edges or thin material layers present in the input (see the parameter descriptions), thereby facilitating a more robust volume meshing. See [3][4][5] for a description of the algorithm.

```
Tools {
   DualContouringSurface {
      decimation = true | false
      minAngle = float
      minCutLength = float
      minDihedralAngle = float
      minDualEdgeLengthFactor = float
      minPrimalSubedgeLengthFactor = float
      resolution = float
      smoothing = true | false
      snapBoundaryToBox = integer
   }
}
```

The following parameters control the quality of the discretization:

- `decimation` applies a postprocessing algorithm to reduce the vertex count on the output surface. Decimation might produce a surface that violates the `minAngle` value, or the `minDihedralAngle` value, or both values. Default: `false`
- `minAngle` is the minimum-allowable surface angle on the output surface. Default: 1°
- `minCutLength` (in µm) is the minimum length between consecutive cuts on a primal edge. Cuts with a distance less than this threshold are merged. This parameter facilitates the collapse of thin features on the output surface. The acceptable range of values is (0, `resolution`). The higher the value, the more aggressive the collapse. Values greater than 0.5×`resolution` might compromise the geometric accuracy of the output surface. Default: 1e-6
- `minDihedralAngle` the minimum-allowable dihedral angle on the output surface. Default: 5°
- `minDualEdgeLengthFactor` is a factor to compute the minimum length of a dual edge (that is, an edge on the output surface) relative to the minimum length of a primal edge (that is, `resolution`). This parameter is used to collapse the short edges on the output surface. The acceptable range of values is (0, 1). The higher the value, the more aggressive the collapse. Default: 0.05
- `minPrimalSubedgeLengthFactor` is a factor to compute the minimum length of a primal sub-edge relative to the minimum length of a primal edge (that is, `resolution`). The acceptable range of values is (0, 1). The higher the value, the more aggressive the collapse of thin features. A value greater than 0.5 might compromise the geometric accuracy of the output surface. Default: 0.05
- `resolution` (in µm) defines the minimum spacing of a primal mesh (that is, the octree grid) used for the reconstruction. The smaller the value, the more accurately the output surface represents the input surface.
- `smoothing` reduces the noise on the output surface. Default: `false`
- `snapBoundaryToBox` snaps the vertices on the exterior output surface to the sides of the bounding box defined by the input surface. Snapping is applied only if gas material is present on the input surface. A value of 0 deactivates snapping. A value of 1 allows snapping to all sides of the box. A value of 2 allows snapping to all sides of the box excluding the +z-side. Default: 0

Note:

The dual-contouring algorithm uses multithreading if you specify `IOControls{numThreads=`*`integer`*`}` (see IOControls Section on page 16).

Generating a Volume Polyhedral Mesh Using the Volume Dual-Contouring Algorithm

Note:

This algorithm is experimental. It is not suitable for applications where a good-quality mesh is required (for example, mechanics).

This subsection generates a volume mesh using the volume dual-contouring algorithm. The mesh contains basic element types (such as tetrahedra, pyramids, prisms, and hexahedra) as well as general polyhedra. The produced boundary faces are triangular. Polygonal faces with more than three vertices are planar.

```
Tools {
   DualContouringVolume {
      resolution = float
   }
}
```

The parameter `resolution` (in μm) defines the minimum spacing of a primal mesh (that is, the octree grid) used for the volume mesh creation. The smaller the value, the more accurately the output surface represents the input surface. A very small value can result in an excessive number of elements and a longer runtime.

Note:

The volume dual-contouring algorithm uses multithreading if `IOControls {numThreads=`*`integer`*`}` is specified. The output polyhedral mesh is written in VTK file format if `IOControls {outputFile= "`*`string`*`"}` is specified. See IOControls Section on page 16.

Interpolating a Source Mesh to a Destination Mesh

This subsection allows interpolation from the source mesh to the destination mesh:

```
Tools {
   InterpolateMesh {
      Conservative
      DstMesh = "string"
      Extrapolate = true | false
      IgnoreMaterials
      Species {"string" "string" ...}
      SrcMesh = "string"
      Tolerance = float
   }
}
```

The following parameters control interpolation:

- `SrcMesh` specifies the file name of the source mesh, and `DstMesh` specifies the file name of the destination mesh. By default, if no species is specified, all the fields in the source mesh are considered for interpolation.
- If you specify the `Conservative` option, then a second-order method described in [7] is used to perform the interpolation.
- When the boundary of the source mesh and the destination mesh do not coincide exactly, `Extrapolate=true` (default) performs the extrapolation by assigning the field value for the destination point from the closest point on the source mesh. For some applications, extrapolation is not wanted, in which case, set `Extrapolate=false` and set the appropriate `Tolerance` for searching for source elements that contain destination points.
- By default, interpolation is performed between two identical materials, but you can use the `IgnoreMaterials` option to override this behavior.
- If any species is specified in the `Species` statement, only those species are interpolated. If the species is already present in the destination mesh, the values are overwritten with new interpolated values.

Performing a 2D Slice of 3D Mesh or Boundary

This subsection performs a 2D slice of a 3D mesh or boundary:

```
Tools {
   Slice {
      location = (float float float)
      normal = (float float float)
   } |
   Slice {
      Direction = X | Y | Z
      Endpoint = (float float)
      Startpoint = (float float)
   }
}
```

Note:

This operation applies to both the mesh and the boundary.

The 2D slice is defined by a plane normal and a location (see Slicing a 3D Mesh Using a Plane and Its Location on page 136).

Alternatively, the slice can be obtained by restricting a plane by a segment. For this interface, you can use the `Direction` parameter to indicate the plane in which the segment lies, and a starting point and an endpoint of a segment are needed. The starting point and

endpoint are represented by only two coordinates of the segment, and the third coordinate is computed from the input structure. For example:

- If the direction is the x-plane, then the coordinates of `Startpoint` and `Endpoint` represent the y- and z-values of the segment.
- If the direction is the y-plane, then the coordinates of `Startpoint` and `Endpoint` represent the x- and z-values of the segment.
- If the direction is the z-plane, then the coordinates of `Startpoint` and `Endpoint` represent the x- and y-values of the segment.

Cutting a Mesh With a Plane

This subsection cuts a mesh with a plane:

```
Tools {
   Cut {
      normal = (float float float)
      location = (float float float)
   }
}
```

The mesh that is to the right of the plane is removed. The right side of the plane is defined as the one to which the normal points.

Note:

This operation is limited to 3D meshes and 3D boundaries. For hybrid meshes, cutting through elements will result in significant topological changes around the cutplane.

Reflecting a Mesh

This subsection mirrors a mesh about a location and appends it to the original mesh:

```
Tools {
   Reflection {
      axis = xmin | ymin | zmin | xmax | ymax | zmax
      map "stringA" = "stringB"
   }
}
```

Note:

This operation applies to both unstructured meshes and tensor-product meshes.

The `map` statement specifies the name that corresponds an input region to the mirrored region. By default, if `map` is not specified, the new region names are given the `_mirrored` file extension.

Extruding a Mesh

The `Extrude` subsection extrudes a planar 2D mesh along a nonuniform 1D grid to generate a 3D mesh. You can extrude a mesh by specifying either `spacingMethod` and `zCuts`, or `extension` and `steps`.

The syntax is:

```
Tools {
   Extrude {
      spacingMethod= even | regular | smooth
      zCuts= (float) | ((float float) ...) | ((float float boolean) ...)
   }
}
```

or:

```
Tools {
   Extrude {
      extension = float
      steps = integer
   }
}
```

The following parameters control extrusion of a mesh:

- `spacingMethod` specifies the type of progression used when a sequence of cutplanes is generated between adjacent pair of planes. Options are:
  - `even` (default) distributes the cuts evenly, trying to approximate the spacing specified at the beginning of the interval.
  - `regular` distributes the cuts evenly, using the exact spacing specified at the beginning of the interval, and leaves the last interval with an approximate size if there is no more room to accommodate the requested size.
  - `smooth` distributes the cuts to have a smooth grading of spacing between planes.
- `zCuts` represents the cutplanes for extrusion. Each cutplane can be defined in the following ways:
  - A single floating-point number represents the distance of the cut parallel to the 2D planar mesh.

- A pair of floating-point numbers, where the first number is the distance of the cut from the 2D mesh, and the second number is the expansion of the cuts into a sequence of planes to be generated between adjacent pair of planes.
- A tuple of float, float, and Boolean, where the first number is the distance of the cut from the 2D mesh, and the second number is the expansion of the cuts into a sequence of planes to be generated between adjacent pair of planes. The Boolean parameter allows a cutplane to be a region boundary for a subregion. By default, this value is `false`.

For example, `zCuts= (0.2 0.5 0.8)` generates three cutplanes at distances 0.2, 0.5, and 0.8 µm from the input 2D mesh.

- `extension` is the extrusion distance along the normal direction of the 2D object.
- `steps` is the number of (equal) subdivisions of the extruded object along the normal direction of the 2D object.

Examples

The following example creates a series of cutplanes at distances 1.2, 2.5, and 2.9 µm from the input 2D mesh:

```
zCuts = ( (1.2 0.3) (2.5 0.2) (2.9) )
```

In this case, between 1.2 and 2.5 µm, additional cutplanes are generated within 0.3 µm intervals. Between 2.5 and 2.9 µm, additional cutplanes are generated within 0.2 µm intervals (depending on the value of the `spacingMethod` parameter).

The following example works similarly to the previous example, with the only difference being that the cutplane at a distance of 2.5 µm is now a subregion boundary:

```
zCuts = ( (1.2 0.3) (2.5 0.2 true) (2.9) )
```

Therefore, the extrusion creates two different subregions: Region1 from z=0 to z=2.5, and Region2 from z=2.5 to z=2.9.

Stretching a Mesh

This subsection stretches an existing mesh by adding a new column of elements at the specified location in the specified direction:

```
Tools {
   Stretch {
      direction = X | Y | Z
      length = float
      location = (float float float)
   }
}
```

The unit of length is the same as that of the input mesh. A negative length indicates a stretch in the opposite direction.

Note:

This operation is limited to meshes.

Remapping Mesh Region Names and Region Materials

This subsection allows you to change the attributes of a mesh region, that is, its name and material:

```
Tools {
   MapRegions {
      RegionChange = (("stringA" "stringB") ...)
      MaterialChange = (("stringA" "stringB") ...)
   }
}
```

The following parameters change the name and material of regions:

- `RegionChange` accepts a list of string–string pairs. For each entry, the first string (`stringA`) represents the existing region name in the mesh. The command changes this region name to the second string (`stringB`). If `stringA` does not represent a valid mesh region, the command has no effect.
- `MaterialChange` accepts a list of string–string pairs. For each entry, the first string (`stringA`) represents the existing region name in the mesh. The command changes the material of this region to the material represented by the second string (`stringB`). If `stringA` does not represent a valid mesh region or `stringB` does not represent a valid DATEX material, the command has no effect.

Placing Individual Dopant of Species

This subsection allows you to place an individual dopant of a species at a specified location:

```
Tools {
   Dopants {
      Species "string" {
         DopantLocation = (float float float)
         ...
         DopantLocation = (float float float)
         Replace
      }
   }
}
```

If the specified location matches the mesh point, the discrete dopant will be assigned to that particular point or it will snap to the nearest mesh point. By default, the specified dopants for each species are added to the input continuous doping. If you specify the `Replace` option, the input continuous doping is reinitialized to zero for a user-specified species only in those regions that contain the specified single dopants. The other species of this region are untouched.

For example, if you specify single dopants of `BoronActiveConcentration` with the `Replace` option, only the input `BoronActiveConcentration` of the region that contains these single dopants is reinitialized to zero, and the contribution from single dopants is considered in that region. The other species of this region are untouched.

Extracting a Boundary From a Mesh

You can specify the `Mesh2bnd` option to extract a boundary from a mesh, or you can set the geometric accuracy and short edge to clean up unwanted small features of the input geometry:

```
Tools {
   Mesh2bnd |

   Mesh2bnd {
      accuracy = float
      shortedge = float
   }
}
```

The default value of `accuracy` is `1e-8` μm. The geometric accuracy cleans up the coplanar mesh points. By default, this operation does not remove short edges.

Converting a Tetrahedral Mesh to a Hybrid Mesh

Hybrid meshes (also referred to as mixed-element meshes) contain hexahedra, prisms, pyramids, and tetrahedra. They are used in some tools such as Sentaurus Device because, compared to tetrahedral meshes, hybrid meshes have fewer elements, thereby allowing the tools to perform simulations on larger structures. Hybrid meshes are also used in Sentaurus Interconnect to increase the accuracy of the simulation. If the input tetrahedral mesh contains contact regions, or interface regions, or both, then they transfer automatically to the hybrid mesh.

The `Mesh2Hybrid` option converts an input mesh containing only tetrahedral elements to a hybrid mesh:

```
Tools {
   Mesh2Hybrid
}
```

Alternatively, you can specify the `Mesh2Hybrid` option directly from the command line by specifying the `-H` option.

Note:

Sentaurus Mesh cannot use hybrid meshes directly as input for any operation specified in the `Tools` section.

When the `Mesh2Hybrid` option is processed, the output is written out directly, thereby ignoring the operations that follow in the `Tools` section.

Specifying an Algorithm for Smoothing Noise

This subsection uses a multimaterial level-set (MLS) algorithm to smooth any noise that might be present in the boundary file:

```
Tools {
   MultiLevelSetBrepFilter {
      CellSize = float
      numThreads = integer
   }
}
```

The output of this operation can contain many poor-quality triangles on non-flat surfaces. So, you can use the `DelaunaySurfaceRemeshing` subsection afterwards to eliminate these triangles (see Rediscretizing the Boundary Using the DelPSC Algorithm on page 73).

The `CellSize` parameter specifies the level-set cell size, which should be, at most, one-third the thickness of the thinnest region. Otherwise, the thin region can be considered noise and it disappears. The coarser the cell size, the more features can be smoothened. The default is 0.001 µm.

The amount of geometry smoothing performed by the MLS algorithm depends on both the curvatures in the input and the level-set cell size. A noisy surface has a high curvature, so it will be smoothed to a large extent to remove noise. On the other hand, a planar surface has zero curvature, so it will be well preserved. Unfortunately, a sharp corner has a theoretically infinite curvature, so it will become a rounded corner. The specified level-set cell size is the threshold to distinguish between the noise to be removed and the features to be preserved.

The `numThreads` parameter specifies the number of threads to use (default is 1). You can use the `--max_threads` command-line option to override the value specified by `numThreads`.

Creating Structures With Randomized Doping Profiles

This subsection creates structures with randomized doping profiles based on an original structure obtained from process simulation or created analytically. It works by *atomizing* the

original continuous doping distribution to create discrete dopants and then reassigning the doping associated with these discrete dopants back to the surrounding mesh nodes. Different atomizations or randomized structures can be obtained from one original structure.

Note:

When the `RandomizeDoping` subsection is processed, the output is written out directly, thereby ignoring the operations that follow in the `Tools` section.

This operation does not change the mesh but reassigns the randomized doping.

The syntax is:

```
Tools {
   RandomizeDoping {
      ContinuousContactDoping
      DopingAssignment = "CIC" | "NGP" | "Sano"
      FileIndex = integer
      NumberOfRandomizedProfiles = integer
      SaveDiscreteDopants

      Material "Material name 1" {
         Species "Dataset name 1" {
            AutoScreeningFactor
            Ignore | Randomize
            ScreeningFactor = value
         }
         Species "Dataset name 2" {
            AutoScreeningFactor
            Ignore | Randomize
            ScreeningFactor = value
         }
         ...
      }

      Material "Material name 2" {
         Species "Dataset name 1" {
            AutoScreeningFactor
            Ignore | Randomize
            ScreeningFactor = value
         }
         Species "Dataset name 2" {
            AutoScreeningFactor
            Ignore | Randomize
            ScreeningFactor = value
         }
         ...
      }

      Region "Region name 1" {
         Species "Dataset name 1" {
            AutoScreeningFactor
            Ignore | Randomize
```

```
            ScreeningFactor = value
         }
         Species "Dataset name 2" {
            AutoScreeningFactor
            Ignore | Randomize
            ScreeningFactor = value
         }
         ...
      }

      Cuboid [ (value value value) (value value value) ] {
         Species "Dataset name 1" {
            AutoScreeningFactor
            ScreeningFactor = value
            ScreeningScalingFactor = value
         }
         Species "Dataset name 2" {
            AutoScreeningFactor
            ScreeningFactor = value
            ScreeningScalingFactor = value
         }
         ...
      }
      ...
   }
}
```

In the `RandomizeDoping` subsection, only materials that are specified in the command file will have their doping randomized. Materials found in the input structure that are not specified in the command file will have their original continuous doping written to the output file. If a species is not specified for a material, it is simply copied to the output without randomizing.

For example, if the input structure contains the material `"Silicon"` with the species `"ArsenicActiveConcentration"` and `"BoronActiveConcentration"`, and the command file only specifies `"BoronActiveConcentration"`, the randomized species `"BoronActiveConcentration"` and original `"ArsenicActiveConcentration"` contribute to the silicon doping in the final output structure.

In addition to materials, randomization can be restricted solely to a region or a cuboid.

Parameters

Default values are given in parentheses if applicable.

`AutoScreeningFactor`

This option and the `ScreeningFactor` parameter apply only when `DopingAssignment= "Sano"`. The `AutoScreeningFactor` calculation is invoked only if specified.

`ContinuousContactDoping`

Specifying this option discards the randomized doping assigned to the contact nodes and, instead, it uses the contact doping obtained from the original structure.

`DopingAssignment ("Sano")`

Specifies the method used to assign doping from the discrete dopants created during atomization to the mesh nodes:

- The cloud-in-cell (`"CIC"`) method distributes the doping of a particle to the vertex nodes of the element in which the particle is located.
- The nearest grid point (`"NGP"`) method assigns the doping of a particle to the nearest mesh node.
- The `"Sano"` method uses a doping function described in Appendix B on page 162 to distribute the doping of a particle to surrounding nodes. If you select `"Sano"`, you must specify `ScreeningFactor` as well.

`FileIndex (0)`

Specifies the name of output files and also is used as a random seed during randomization. The names of output files are created automatically using the following convention:

```
<root>_<DopingAssignment><FileIndex + Randomized_Profile_Number>
_msh.tdr
```

where `<root>` is a base name of the input TDR file.

For example, if `DopingAssignment= "Sano"`, `FileIndex=1000`, and `NumberOfRandomizedProfiles=3`, and the command line is:

```
snmesh nmos
```

then the following output files are created:

```
nmos_sano1000_msh.tdr
nmos_sano1001_msh.tdr
nmos_sano1002_msh.tdr
```

`Ignore | Randomize`

Two choices are provided for each specified species. By default, all specified species are randomized. With the `Ignore` option, a species is not randomized and is not copied to the output file. All species that are not specified in the command file for a specified material are copied to the output file.

`NumberOfRandomizedProfiles (1)`

Specifies the number of randomized profiles to be generated.

`SaveDiscreteDopants`

This option saves active discrete dopants along with the randomized dopant profiles. The active discrete dopants can be visualized as well as the mesh. This file also can be specified as a `ParticleFile` in the particle profile section of the `Definitions` section (see Defining Particle Profiles on page 24).

`ScreeningFactor`

The `ScreeningFactor` parameter and the `AutoScreeningFactor` option apply only when `DopingAssignment= Sano"`. If `"Sano"` is selected, `ScreeningFactor` must be specified.

`ScreeningScalingFactor`

Controls the degree of smoothness of the profile. It is applied to the screening factor when `AutoScreeningFactor` is specified.

Adding or Removing Interfaces From a Mesh

Some TCAD Sentaurus tools produce meshes that contain an explicit description of the interface between adjacent regions. This description is not understood by some tools and is not produced by other tools, so it is sometimes necessary to add or remove it from a mesh file.

Specify either `addInterfaceRegions` or `removeInterfaceRegions` to add interfaces or to remove interfaces, respectively.

The syntax is:

```
Tools { addInterfaceRegions | removeInterfaceRegions }
```

Generating Voronoï Diagrams

This subsection generates a 3D Voronoï diagram that is constrained at the boundary. It is useful for generating a brep that simulates a polycrystalline grain device, wherein each grain is represented by a unique region in the brep.

The Voronoï sites are inserted by first dividing the brep domain into tensor grids, where the grid spacing is a user-defined parameter. Next, the generator populates a Voronoï site within each grid. This vertex is generated randomly within a grid with some padding. Then, the generator creates a Voronoï diagram and constrains it at the boundary of the input brep.

The syntax is:

```
Tools {
   Voronoi {
      accuracy = float
      brepIn = "string"
```

```
      brepOut = "string"
      siteSpacing = float
   }
}
```

The following parameters control generation of Voronoï diagrams:

- `accuracy` is used to merge vertices that are generated too close to each other in the Voronoï diagram. A sphere of radius `accuracy` is created around a particular vertex in the Voronoï diagram. Any other vertices in the Voronoï diagram that lie within this sphere are merged.
- `brepIn` is the input boundary file, which denotes the boundary inside which you want to create the Voronoï diagram.
- `brepOut` is the output file where the results are stored.
- `siteSpacing` specifies the grid spacing of randomly inserted Voronoï sites.

Example: Simple Cube Geometry

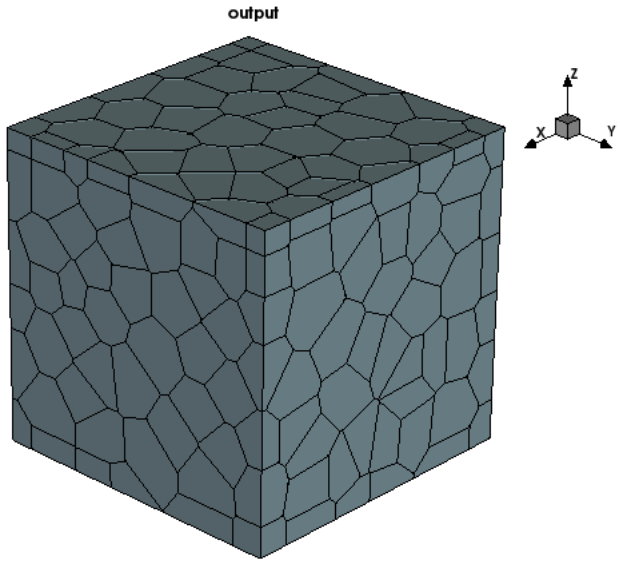
The following example demonstrates the `Tools` section of the command file for a simple cube geometry:

```
Tools {
   Voronoi {
      brepIn = "cube.tdr"
      brepOut = "output.tdr"
      siteSpacing = 0.2
      accuracy = 1e-5
   }
}
```

The Voronoï generator generates random grid-spaced Voronoï sites within the specified input brep by using the value of the `siteSpacing` parameter as the grid spacing.

While generating the Voronoï diagram, all points that lie within the accuracy distance from an already existing point are merged. Figure 4 shows the output.

Figure 4 Constrained 3D Voronoï diagram (output.tdr) with a cube boundary and 695 grains



QualityReport Section

This optional section is used to specify mesh quality limits for mesh generation. Sentaurus Mesh produces a report regardless of whether the limits are satisfied or not. This section of the command file can help to ensure that the mesh is suitable for device simulation.

Note:

The `QualityReport` section applies only to 3D axis-aligned meshes and 3D offsetting meshes. The specified limits are used only to report on the mesh quality and do not affect how meshes are generated.

If any limits are not satisfied, then Sentaurus Mesh saves additional field variables in the output `_msh.tdr` file:

- `AngleElements`: Angle of an element as defined by the box method
- `DelaunayInsphere3D`: Number of elements that are non-Delaunay elements
- `ElementsPerVertex`: Number of elements that share a vertex
- `ElementVolume`: Volume of an element
- `ShortestEdge`: Length of the shortest edge of an element

Syntax of QualityReport Section

```
QualityReport {
   Global
   Material = {stringList}
   Region = {stringList}
   {
      limitMaxConnectivity = integer
      limitMaxNonDelaunay = float
      limitMinAngle = float
      limitMinEdgeLength = float
      limitMinVolume = float
   }
}
```

Parameters

Default values are given in parentheses if applicable.

`Global`

If specified, the limits are evaluated on the entire mesh.

`Material`

If specified, the limits are evaluated on the list of materials only.

`Region`

If specified, the limits are evaluated on the list of regions only.

`limitMaxConnectivity (0)`

Specifies the maximum number of elements that can share any vertex. If this limit is exceeded, then Sentaurus Mesh saves the `ElementsPerVertex` field variable in the output mesh file. The default value of `0` means this parameter has no effect.

`limitMaxNonDelaunay (100.0)`

Specifies the maximum percentage of all elements that can be non-Delaunay elements. If this limit is exceeded, then Sentaurus Mesh saves the `DelaunayInsphere3D` field variable in the output mesh file.

`limitMinAngle (0.0)`

Specifies the minimum angle (given in degrees), defined using the box method, of any element. If this limit is exceeded, then Sentaurus Mesh saves the `AngleElements` field variable in the output mesh file. See *Utilities User Guide*, AngleElements.

`limitMinEdgeLength (0.0)`

Specifies the minimum edge length (in μm) of any element. If this limit is exceeded, Sentaurus Mesh saves the `ShortestEdge` field variable in the output mesh file.

`limitMinVolume (0.0)`

Specifies the minimum volume (in μm^3) of any element. If this limit is exceeded, Sentaurus Mesh saves the `ElementVolume` field variable in the output mesh file.

Examples

Generate a report on the mesh quality of the entire mesh using the default limits:

```
QualityReport {
   Global
}
```

Generate a report on the mesh quality of the entire mesh using the default limits, followed by a report on the mesh quality of the materials `Silicon` and `Oxide` using the default limits:

```
QualityReport {
   Global
   Material = {"Silicon" "Oxide"}
}
```

Generate a report on the mesh quality of the entire mesh using the default limits, followed by a report on the mesh quality of the regions `Substrate` and `Oxide_1` using the default limits:

```
QualityReport {
   Global
   Region = {"Substrate" "Oxide_1"}
}
```

Generate a report on the mesh quality of the entire mesh with two specific limits:

```
QualityReport {
   Global {
      limitMaxNonDelaunay = 0.1
      limitMinAngle = 1e-2
   }
}
```

Generate a report on the mesh quality of the entire mesh with one set of limits, followed by a report on the mesh quality of the material `Silicon` with a different set of limits:

```
QualityReport {
   Global {
      limitMaxNonDelaunay = 0.1
      limitMinAngle = 1e-2
   }
   Material = {"Silicon"} {
      limitMinVolume = 1e-18
      limitMinEdgeLength = 1e-5
   }
}
```

Structured Section

This section specifies the parameters required for the generation of a 2D structured mesh of a Josephson junction. When this section is specified, Sentaurus Mesh calls the structured mesh generator. The `Structured` section also contains the following subsections:

- The `Insulator` subsection specifies the parameters necessary for generating the central insulator region, which has a 2D rectilinear mesh.
- The `LeftSuperConductor` subsection specifies the parameters necessary for generating the left superconductor region. This region has a 2D elliptical mesh such that its refinement decays when moving away from the insulator
- The `RightSuperConductor` subsection specifies the parameters necessary for generating the right superconductor region, which might or might not have parameters like its left counterpart.

The structured mesh generator automatically creates extra spacing beyond the superconductor regions. This spacing is called the *virtual lead region*.

Syntax of Structured Section

```
Structured {
   CenterX = float
   CenterY = float
   ySpacing = integer
   Insulator {
      xSpacing = integer
      height = float
      width = float
      RegionName = string
      MaterialName = string
   }
   LeftSuperConductor {
      xSpacing = integer
      height = float
      width = float
      RegionName = string
      MaterialName = string
      ellipseRatio = float
   }
   RightSuperConductor {
      xSpacing = integer
      height = float
      width = float
      RegionName = string
      MaterialName = string
      ellipseRatio = float
   }
}
```

General Parameters

Default values are given in parentheses if applicable.

`CenterX (0)`

Specifies the x-coordinate of the center of the device.

`CenterY (0)`

Specifies the y-coordinate of the center of the device.

`ySpacing (1)`

Specifies the number of refinement spacings required in the y-direction. This value must be a power of 2 because of the binary splitting nature of refinement.

Parameters for Insulator Subsection

Default values are given in parentheses if applicable.

`xSpacing (1)`

Specifies the number of refinement spacings required in the x-direction for the insulator. This value must be a power of 2 because of the binary splitting nature of refinement.

`height (0)`

Sets the height of the insulator part of the device.

`width (0)`

Sets the width of the insulator part of the device.

`RegionName ("")`

Specifies the name of the insulator region.

`MaterialName("")`

Specifies the material of the insulator region.

Parameters for LeftSuperConductor and RightSuperConductor Subsections

Default values are given in parentheses if applicable.

`xSpacing (1)`

Specifies the number of refinement spacings required in the x-direction for the left or right superconductor region. This value must be a power of 2 because of the binary splitting nature of refinement.

`height (0)`

Sets the height of the left or right superconductor part of the device.

`width (0)`

Sets the width of the left or right superconductor part of the device.

`RegionName ("")`

Specifies the name of the left or right superconductor region.

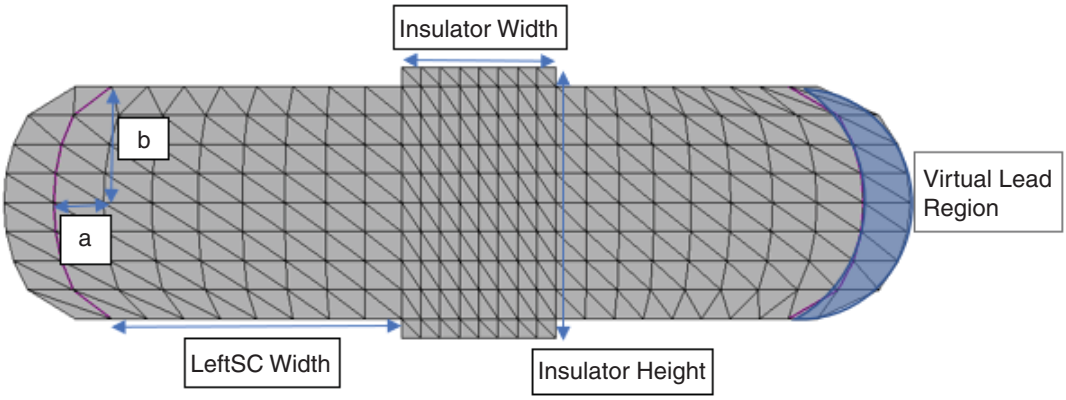
`MaterialName("")`

Specifies the material of the left or right superconductor region.

`ellipseRatio (1)`

Specifies the ratio of the semi-minor axis in the x-direction to the semi-major axis in the y-direction at the extremity of the left or right superconductor.

Figure 5 Example of Josephson junction mesh: a = width and b = height of left superconductor region, with the ellipse ratio = a/b



References

[1] S.-W. Cheng, T. K. Dey, and J. A. Levine, “A Practical Delaunay Meshing Algorithm for a Large Class of Domains,” in *Proceedings of the 16th International Meshing Roundtable*, Seattle, WA, USA, pp. 477–494, October 2007.

[2] T. K. Dey and J. A. Levine, “Delaunay Meshing of Piecewise Smooth Complexes without Expensive Predicates,” *Algorithms*, vol. 2, no. 4, pp. 1327–1349, 2009.

[3] T. Ju *et al*., “Dual Contouring of Hermite Data,” *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 339–346, 2002.

[4] S. Schaefer and J. Warren, *Dual Contouring: “The Secret Sauce”*, Technical Report, Department of Computer Science, Rice University, Houston, TX, USA, March 2003.

[5] Y. Zhang and J. Qian, “Dual Contouring for domains with topology ambiguity,” *Computer Methods in Applied Mechanics and Engineering*, vol. 217–220, pp. 34–45, April 2012.

[6] For information about Neper, a third-party tool that generates 2D and 3D polycrystalline structures with very large number of grains, go to https://neper.info.

[7] F. Alauzet and M. Mehrenberger, “P^1-conservative solution interpolation on unstructured triangular meshes,” *International Journal for Numerical Methods in Engineering*, vol. 84, no. 13, pp. 1552–1588, 2010.

3

Doping and Refinement Examples

This chapter illustrates how to use the Definitions and Placements sections of the command file, as well as how to use the Offsetting section to generate layered meshes in Sentaurus Mesh.

Simple Diode

This section describes the command file `diode.cmd` that is used as an example for the rest of this chapter. The command file contains two types of information: dimension-independent data and dimension-dependent data. The dimension-independent part of the command file `diode.cmd`, for this example, is:

```
Title "minimal example: simple diode"
Definitions {
   # Profiles
   Constant "n-type region" {
      Species = "PhosphorusActiveConcentration" Value = 1e+18
   }
   Constant "p-type region" {
      Species = "BoronActiveConcentration" Value = 1e+17
   }
}
```

The optional keyword `Title` is used for a short description of the device and mesh. The `Definitions` section specifies the dimension-independent part of the command file and can be used for all dimensions without modifications.

Two constant profiles for doping are described using the keyword `Constant` followed by the profile name in double quotation marks. The parameter `Species` specifies the doping species used in the region. The constant concentration is specified by the parameter `Value`. The sign is intrinsic to the species.

Now, the doping profiles must be placed in the device. The placement of these profiles depends on the device geometry.

Since, for this example, ‘solid’ regions are to be filled with constant doping, the following instructions are added to the command file:

```
Placements {
   # Profiles
   Constant "n-type region instance" {
      Reference = "n-type region"
      EvaluateWindow {
         Element = cuboid [(1 0 0), (2 3 2)]   # for three dimensions
      }
   }
   Constant "p-type region instance" {
      Reference = "p-type region"
      EvaluateWindow {
         Element = cuboid [(0 0 0), (1 3 2)]   # for three dimensions
      }
   }
}
```

The keyword `Placements` starts the dimension-dependent section where the instances of the definitions given in the `Definitions` section are defined. The `Reference` parameter specifies a profile defined in the `Definitions` section. The `EvaluateWindow` defines the valid domain for the profiles. In this example, the valid domains are lines in one dimension, rectangles in two dimensions, and cuboids in three dimensions. If `EvaluateWindow` is not defined in the file, the profile is valid in the entire domain of the device.

For the 3D case, the valid domain of the p-type region is the lower half of the device, given by the cuboid [(0 0 0), (1 2 3)]. In two dimensions, this domain is given by the rectangle [(0 0), (1 2)] and in one dimension, by the line [(0), (1)]. However, the doping profile defined for three dimensions can be used for the lower dimensions. For the rest of this chapter, only the command file for the 3D case will be used.

In the example, there is an abrupt decay function between the two constant profiles. The doping associated with points outside the `EvaluateWindow` is zero. This situation can be modified if the parameter `DecayLength` is used. By setting `DecayLength` in `EvaluateWindow`, an error function can be used as a decay profile.

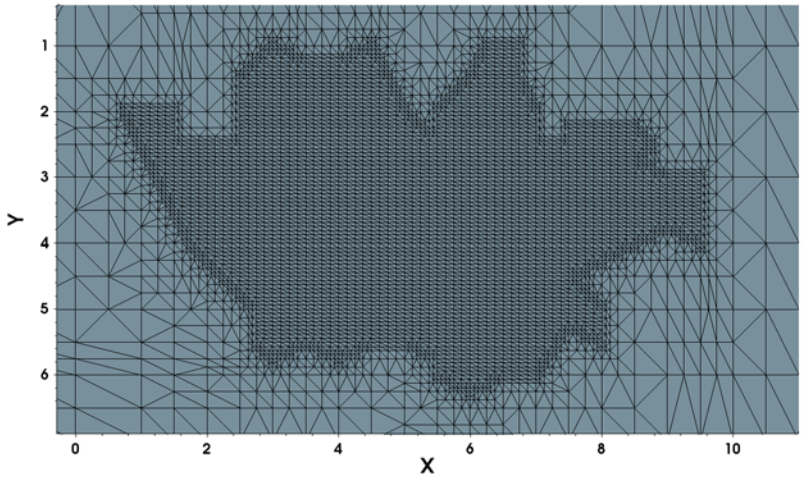
Refinement and Evaluation Windows

The refinement conditions specified in a `Refinement` statement can be restricted by using refinement windows, which can be simple rectangles, polygons, polyhedra, regions, or materials.

Using Refinement Polygons

Figure 6 illustrates the use of polygonal domains for specifying a polygonal `RefineWindow` and for using a polygonal domain as an `EvaluateWindow`.

Figure 6 Polygonal refinement



The domain is a simple rectangular boundary and the command file is:

```
Title "Refinement Polygon"
Definitions {
   Refinement "global" {
      MaxElementSize = (4, 4)
      MinElementSize = (.04 .04)
      RefineFunction = MaxTransDiff(Variable="DopingConcentration",
                                    Value=0.5)
   }
   Refinement "refpol" {
      MaxElementSize = (0.3 0.1)
   }
   Constant "bor" {
      Species = "BoronConcentration" Value=1e+17
   }
}

Placements {
   Refinement "global" {
      Reference = "global"
      RefineWindow = rectangle [( -2 -2 ), ( 14 14 )]
   }
   Refinement "refpol" {
      Reference = "refpol"
      RefineWindow = polygon [( 1 2 ) ( 0.75 2 ) ( 1 2.5 ) ( 1.25 3 )
         ( 1.5 3.5 ) ( 1.75 4 ) ( 2 4.25 ) ( 2.25 4.5 ) ( 2.5 4.75 )
```

```
          ( 2.75 5 ) ( 2.75 5.5 ) ( 3 5.75 ) ( 3.5 5.5 ) ( 4 5.75 )
          ( 4.5 5.5 ) ( 5 5.5 ) ( 5.5 5.75 ) ( 5.5 6 ) ( 6 6.25 )
          ( 6.5 6 ) ( 7 6 ) ( 7.5 5.25 ) ( 8 5.5 ) ( 8 5 ) ( 7.5 4.5 )
          ( 8 4.25 ) ( 8.5 4 ) ( 9 3.75 ) ( 9.5 4 ) ( 9.5 3.5 )
          ( 9.5 3 ) ( 9 3 ) ( 8.5 2.75 ) ( 8.75 2.5 ) ( 8.5 2.25 )
          ( 8 2.25 ) ( 7.5 2.25 ) ( 7.5 2.5 ) ( 7 2.5 ) ( 7 2 )
          ( 6.75 1.5 ) ( 6.75 1 ) ( 6.25 1  ) ( 6 1.5 ) ( 5.5 2  )
          ( 5.5 2.5 ) ( 5 2 ) ( 4.75 1.5 ) ( 4.5 1 ) ( 4 1.25 )
          ( 3.5 1.25 ) ( 3 1 ) ( 2.5 1.5 ) ( 2.5 2 ) ( 2.5 2.5 )
          ( 2 2.5 ) ( 1.5 2.5 ) ( 1.5 2 ) ( 1 2 )]
   }
   Constant "bor" {
      Reference = "bor"
      EvaluateWindow {
           Element = polygon [( 1 2 ) ( 0.75 2 ) ( 1 2.5 ) ( 1.25 3 )
           ( 1.5 3.5 ) ( 1.75 4 ) ( 2 4.25 ) ( 2.25 4.5 )
           ( 2.5 4.75 ) ( 2.75 5 ) ( 2.75 5.5 ) ( 3 5.75 ) ( 3.5 5.5 )
           ( 4 5.75 ) ( 4.5 5.5 ) ( 5 5.5 ) ( 5.5 5.75 ) ( 5.5 6 )
           ( 6 6.25 ) ( 6.5 6 ) ( 7 6 ) ( 7.5 5.25 )
           ( 8 5.5 ) ( 8 5 ) ( 7.5 4.5 ) ( 8 4.25 ) ( 8.5 4 ) ( 9 3.75 )
           ( 9.5 4 ) ( 9.5 3.5 ) ( 9.5 3 ) ( 9 3 ) ( 8.5 2.75 )
           ( 8.75 2.5 ) ( 8.5 2.25 ) ( 8 2.25 ) ( 7.5 2.25 )
           ( 7.5 2.5 ) ( 7 2.5 ) ( 7 2 ) ( 6.75 1.5 ) ( 6.75 1 )
           ( 6.25 1 ) ( 6 1.5 ) ( 5.5 2 ) ( 5.5 2.5  ) ( 5 2  )
           ( 4.75 1.5 ) ( 4.5 1 ) ( 4 1.25 ) ( 3.5 1.25 ) ( 3 1 )
           ( 2.5 1.5 ) ( 2.5 2 ) ( 2.5 2.5 ) ( 2 2.5 ) ( 1.5 2.5 )
           ( 1.5 2 )( 1 2 )
      }
   }
}
```

Using Composite Elements

You can combine geometric elements to form more complex elements called *composite elements*. These elements can be used to define curved reference elements for analytic profiles, which otherwise cannot be correctly defined using standard elements. An example of the use of a composite element is:

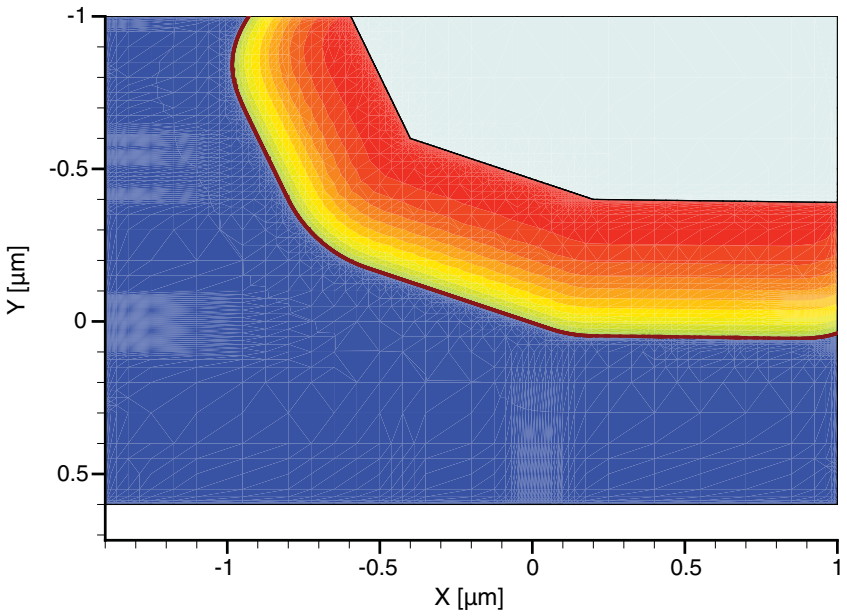
```
AnalyticalProfile "MyProfile" {
   Reference = "MyProfileReference"
   ReferenceElement {
      Element = {
         line [( -1 2 ) , ( 4 2 )]
         line [( 4 2 ) , ( 6 4 )]
         line [( 6 4 ) , ( 7 4 )]
      }
   }
}
```

Note:

For composite elements to be effective, all components must be adjacent to each other, without gaps between them.

Composite elements are available only in Sentaurus Mesh.

Figure 7 Profile built by combining several reference elements

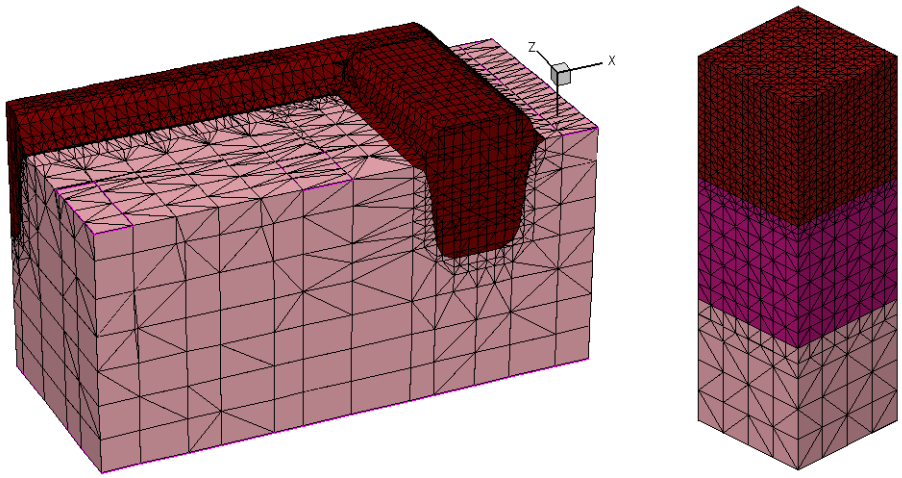


Regionwise and Materialwise Refinement

Figure 8 illustrates the effect of using regionwise and materialwise refinement. The following excerpt shows the relevant part of the command file:

```
Placements {
   Refinement "A" {
      Reference = "A"
      RefineWindow = region ["Ox_Region"]
   }
   Refinement "B" {
      Reference = "B"
      RefineWindow = material ["Oxide"]
   }
}
```

Figure 8 (Left) Regionwise and (right) materialwise refinement

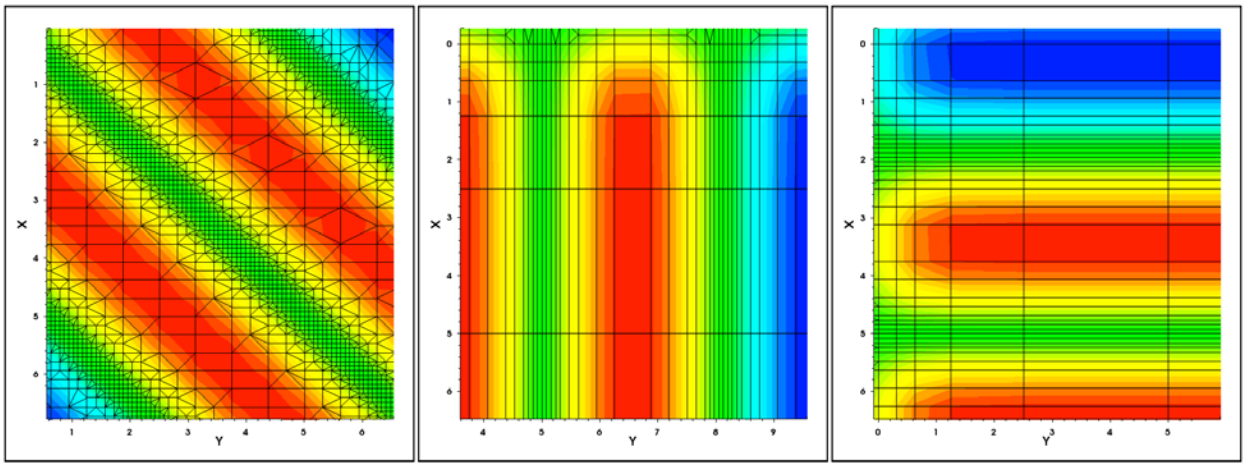


Using Analytic Functions for Doping Specification

This example illustrates the use of general analytic functions for defining doping profiles. To use the primary and lateral directions as x and y, the keyword `Eval` must be specified (instead of `General`), that is, by using global spatial coordinates. Figure 9 shows the generated meshes.

```
AnalyticalProfile "NoName_0" {
   Species = "DopingConcentration"
   Function = Eval(init="a=10",function = "a*sin(x)*cos(y)",value = 0)
}
ReferenceElement {
   Element = line [( 0 0 ), ( 10 10 )]
   # Element = line [( 0 0 ), ( 10 0 )]
   # Element = line [( 0 0 ), ( 0 10 )]
}
```

Figure 9 *Use of analytic refinement functions for doping, where the Element lines in the control block are defined as (left) line [(0 0), (10 10)], (middle) line [(0 0), (10 0)], and (right) line [(0 0), (0 10)]*



Creating 3D Profiles From 2D Cross Sections

A 2D profile can be extended into three dimensions by using an `EvaluateWindow` containing a `sweepElement`, which is an advanced type of element that allows a 2D geometric element to be either swept along a path or swept about a reference axis.

A `sweepElement` consists of a base element, which can be either a polygon or a rectangle, and a path, which can be represented in several ways. When a 3D profile is evaluated, the `sweepElement` takes the 3D coordinate and follows the path in reverse, calculating a local 2D coordinate on the base element. This 2D coordinate is then used to evaluate the 2D profile and to provide the values in three dimensions.

Note:

A `sweepElement` can be used only in `EvaluateWindow` statements. Using it in a `ReferenceElement` or `RefinementWindow` statement generates error messages.

To sweep a 2D profile along a path, use the one of the following syntax examples:

```
# Sweep the base element a distance along the normal to the element.
sweepElement {
   base = <2D Element>
   distance = <double>
}

# Sweep the base element along a vector. The vector must be
# normal to the base element.
sweepElement {
   base = <2D Element>
   vector = (x1,y1,z1)
```

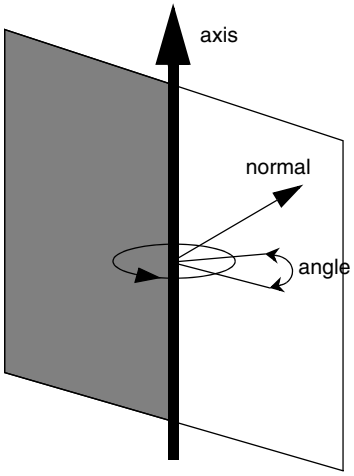
```
}

# Sweep the base element along a polygonal path. The origin of
# the path must be normal to the base element.
sweepElement {
   base = <2D Element>
   path = [(x1,y1,z1)...(xn,yn,zn)]
}

# Rotate the base element about an axis parallel to the z-axis.
# The axis will be placed at the center of the base element.
sweepElement {
   base = <2D Element>
   angle = <double>
}

# Rotate the base element about an axis parallel to the z-axis
# which is placed at a point "p".
sweepElement {
   base    = <2D Element>
   point   = p
   angle   = <double>
}

# Rotate the base element about an axis.
sweepElement {
   base    = <2D Element>
   point   = (x,y,z)
   axis    = (dx,dy,dz)
   angle   = <double>
}
```

In some cases, the normal of the base element is used to determine the direction of the sweep. This normal is calculated in the following way:

- For a polygonal base element that is described by using the sequence $[p_1,p_2,...,p_n]$, the normal is calculated as $(p_3 - p_2)*(p_1 - p_2)$.
- For a rectangular base element described by $[p_1,p_2]$, the normal calculation is extremely complicated and is not explained here. The recommendation is to use polygonal elements, or to swap p_1 and p_2 if the profile is being produced in the wrong direction.

More considerations arise when rotating an element about an axis (see Figure 10 on page 104).

Figure 10 Rotation of a profile about an axis; the gray portion is ignored during the sweep



Since there are some degrees of freedom to perform the rotation, additional rules must be applied:

- Only the right side of the profile is used in the sweep. This avoids double-defined values, which occur when the rotation axis is contained inside the base element and the rotation angle is more than 180°.
- The direction of the rotation is adjusted to match the direction of the normal.

You can set the angle to a negative value, or reverse the orientation of the axis, to obtain the required profile.

When a 2D submesh profile is loaded into a 3D simulation, the default is to place it along the xy coordinate plane. The `ShiftVector` and `Rotation` parameters must be used to place the profile at the required location.

For example, to place a submesh on the xz plane, passing through the point (0, 50, 50), and then to sweep it along a path, the following code can be used:

```
SubMesh "profile" {
   Reference = "SubMesh"
   Rotation {
      angle = -90
      axis = X
   }
   ShiftVector = (0 50 50)
   EvaluateWindow {
      Element = sweepElement {
         base = rectangle [(0 45 0), (15 45 50)]
         path = [(15 45 0) (15 35 0) (30 45 0) (35 25 0)]
      }
      DecayLength = 1
```

```
    }
}
```

Creating Analytic Profiles and Meshes for Grain Boundaries

This section provides examples of creating different analytic profiles and meshes for grain boundaries.

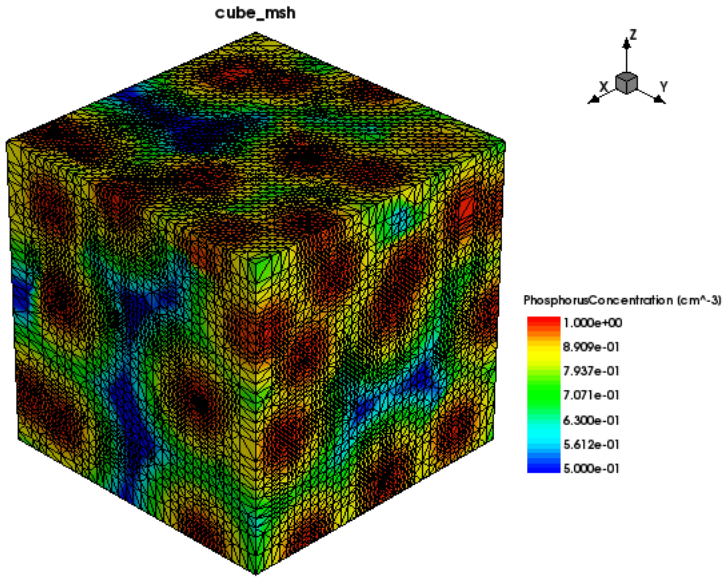
Using GeoPointCloud Element and MaxLenPointCloud Refinement

This example demonstrates a simple cube geometry. The `traps.txt` file generated in the Example: Simple Cube Geometry on page 70 is used as input to the `MaxLenPointCloud` refinement and the `GeoPointCloud` element. The `MaxLenPointCloud` function generates refinement on the basis of the distance of a query edge from its nearest point cloud entry. This point cloud is supplied with the `fileName` parameter.

The analytic profile is specified using the `GeoPointCloud` geometric element. This element accepts a point cloud supplied using the `fileName` parameter. This field has its peak value at a point cloud entry, and it will continue to decay as it moves away from that point cloud entry.

```
Definitions {
   Refinement "Refinement" {
      MaxElementSize = (0.2 0.2 0.2)
      MinElementSize = (0.01 0.01 0.01)
      RefineFunction = MaxLenPointCloud(fileName = "traps.txt",
                                        value = 0.025, factor = 1.2)
   }
   AnalyticalProfile "Traps" {
      Species = "PhosphorusConcentration"
      Function = Gauss(PeakPos = 0, PeakVal = 1, StdDev = 0.5)
      LateralFunction = Gauss(Factor = 0.0)
   }
}

Placements {
   Refinement "Refinement" {
      Reference = "Refinement"
   }
   AnalyticalProfile "Traps" {
      Reference = "Traps"
      ReferenceElement {
         Element = GeoPointCloud{"traps.txt"}
      }
   }
}
```

Figure 11 Trap delineation by using GeoPointCloud element and MaxLenPointCloud function



Using GeoBrep Element and MaxLenInt Refinement

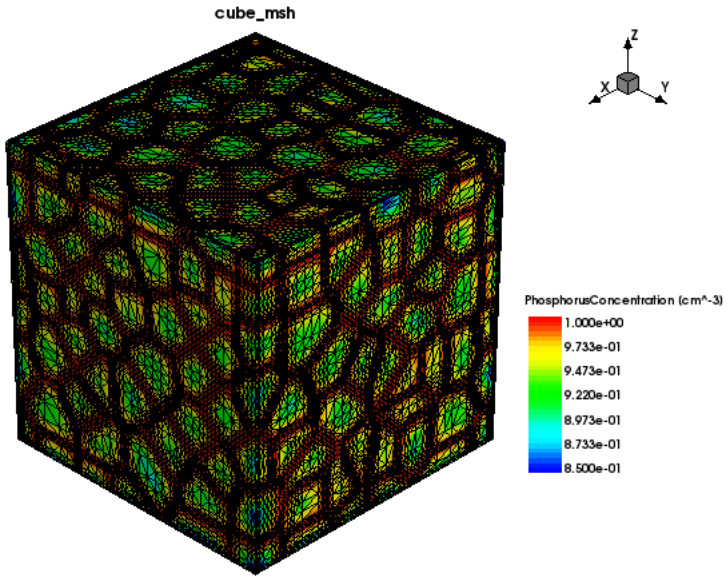
This example demonstrates a simple cube geometry. The `output.tdr` file generated in Example: Simple Cube Geometry on page 88 is used as input for the `MaxLenInt` refinement and the `GeoBrep` element.

```
Definitions {
   Refinement "Refinement" {
      MaxElementSize = (0.2 0.2 0.2)
      MinElementSize = (0.01 0.01 0.01)
      RefineFunction = MaxLenInt(Interface("Silicon", "All"),
                        Value = 0.025, doubleside, factor = 1.2,
                        brep = "output.tdr")
   }
   AnalyticalProfile "buried n-channel" {
      Species = "PhosphorusConcentration"
      Function = Gauss(PeakPos = 0, PeakVal = 1, StdDev = 0.25)
      LateralFunction = Gauss(Factor = 0.3)
   }
}

Placements {
   Refinement "Refinement" {
      Reference = "Refinement"
   }
   AnalyticalProfile "buried n-channel" {
      Reference = "buried n-channel"
      ReferenceElement {
         Element = GeoBrep{"output.tdr"}
```

```
        }
    }
}
```

The `MaxLenInt` function with the `brep` parameter generates refinement along the interfaces of the brep. This refinement is useful for simulating transport phenomena along the grain boundaries. Figure 12 shows the output.

Figure 12 Grain field (analytic) profile with refinement at grain boundaries



Generating Nonvolatile Memory Devices

This example illustrates how to use geometric elements and refinements to generate nonvolatile memory devices.

The command file is presented here:

```
Title "Nonvolatile Memory Example"
Definitions {
   AnalyticalProfile "GB" {
      Species= "PMIUserField0"
      Function= Gauss(PeakPos= 0, PeakVal= 1, StdDev= 0.01)
      LateralFunction= Gauss(Factor= 0.1)
   }

   Refinement "Granularization" {
      RefineFunction= MaxLenInt(Interface("Silicon","All"),
                        Value= 0.01, doubleside, factor= 2.0,
                        brep= "n16_bnd.tdr")
```

```
      }
      # Generate an interface-based implicit refinement by specifying the
      # brep parameter in MaxLenInt. This brep is the grain structure
      # (see Figure 15 (left)).

      AnalyticalProfile "DiscreteTraps" {
         Species= "DeepLevels"
         Function= Gauss(PeakPos= 0, PeakVal= 1, StdDev= 0.001)
         LateralFunction= Gauss(Factor= 0.0)
      }

      Refinement "Traps" {
         RefineFunction= MaxLenPointCloud(fileName= "n20_traps.txt",
                           Value= 0.001, Factor= 1.2)
      }
      # Generate an interface-based implicit refinement by specifying a
      # point cloud file in CSV format (see Figure 15 (right).
   }

   Placements {
      AnalyticalProfile "GB" {
         Reference= "GB"
         ReferenceElement {
            Element= GeoBrep{"n16_bnd.tdr"}
         }
         # Use the GeoBrep geometric element to place the analytic profile
         # along a brep such that the peak value of the analytic profile
         # is at the brep interfaces and it decays as you move away
         # from the interfaces (see Figure 14 (left)).

         EvaluateWindow {
            Element= region ["R.Channel"]
         }
      }

      Refinement "Granularization" {
         Reference= "Granularization"
         RefineWindow = region ["R.Channel"]
      }

      AnalyticalProfile "DiscreteTraps" {
         Reference= "DiscreteTraps"
         ReferenceElement {
            Element= GeoPointCloud{"n20_traps.txt"}
         }
         # Use the GeoPointCloud geometric element to place the analytic
         # profile along the points of a point cloud such that the peak
         # value of the analytic profile is at the point locations and it
         # decays as you move away from the interface
         # (see Figure 14 (right)).

         EvaluateWindow {
            Element= region ["R.Channel"]
```

```
        }
    }

    Refinement "Traps" {
        Reference= "Traps"
        RefineWindow= region ["R.Channel"]
    }
}
```

Figure 13 (Left) Complete nonvolatile device and (right) grain boundary

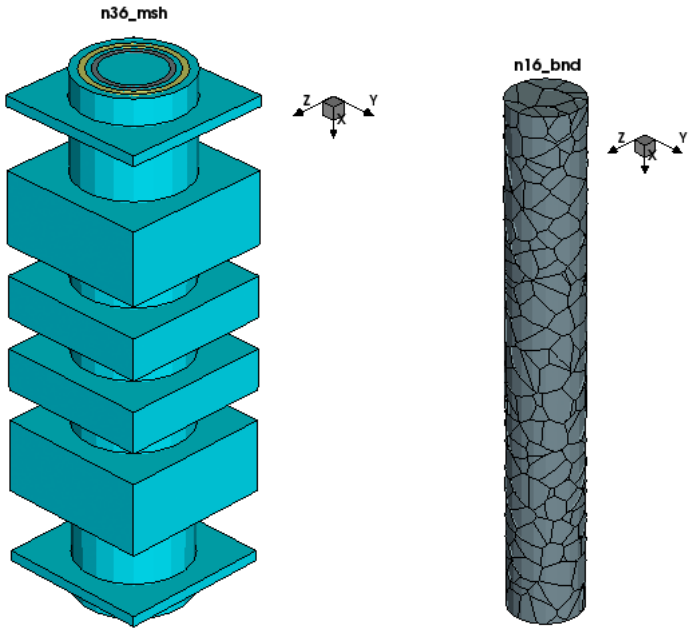


Figure 14 Analytic profile placed using (left) GeoBrep geometric element and (right) GeoPointCloud geometric element

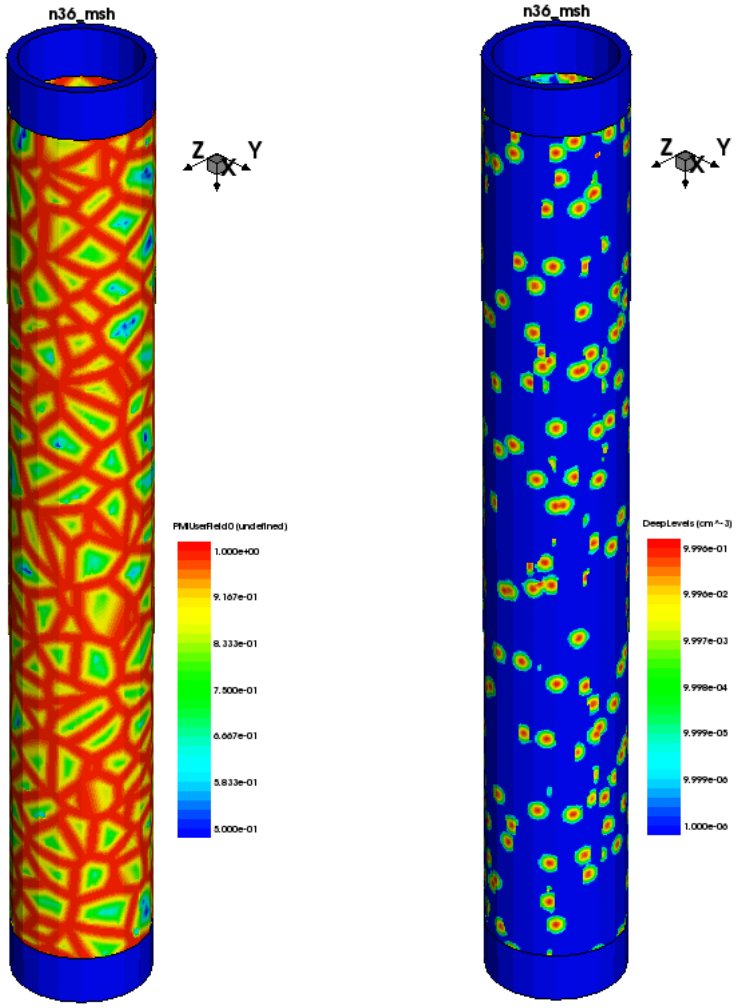
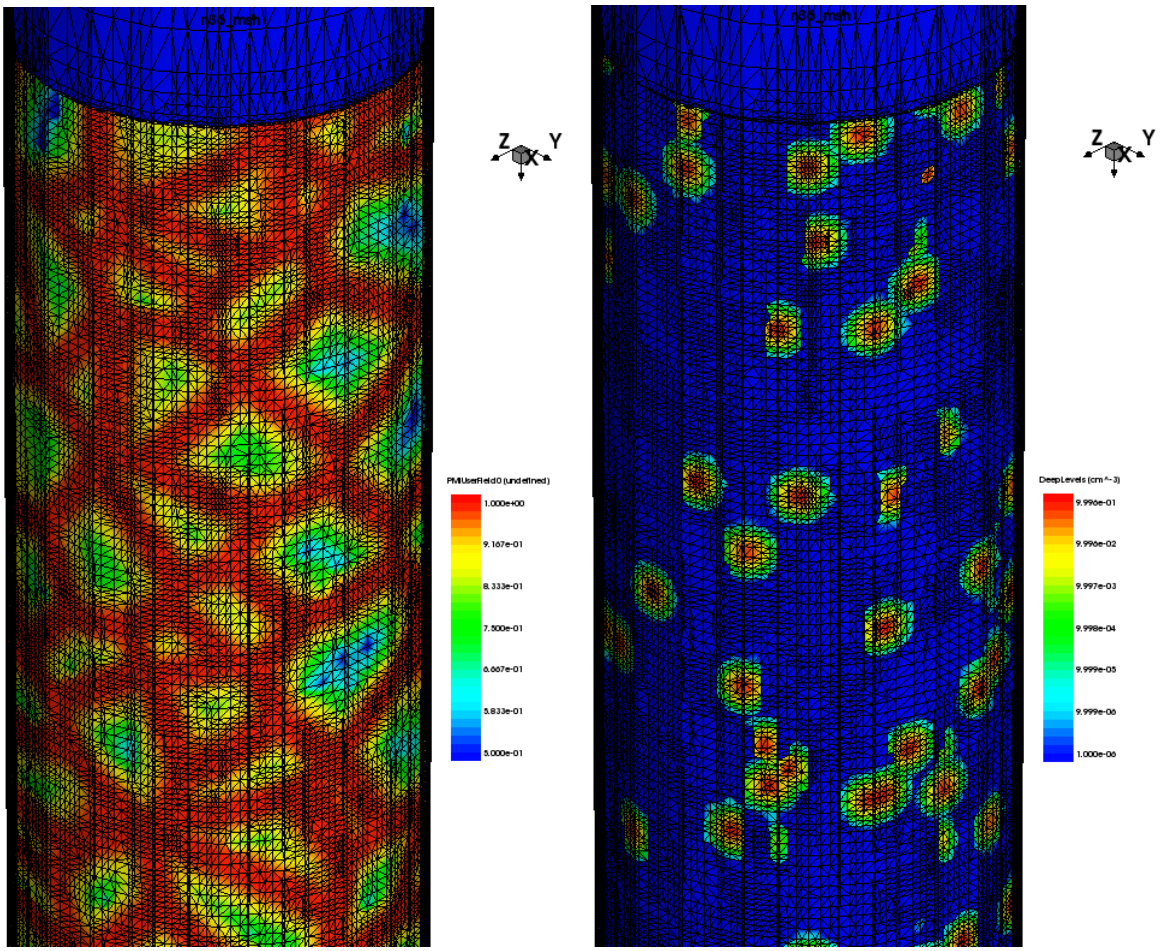


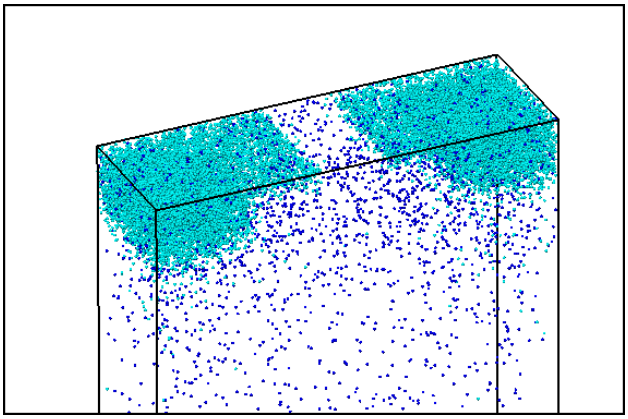
Figure 15 Refinement using (left) MaxLenInt function with brep parameter and (right) MaxLenPointCloud function



Using Particle Profiles to Specify Doping

This example illustrates the use of particle profiles for specifying the doping for a 30-nm n-channel MOSFET. The particle information is generated by Sentaurus Process Kinetic Monte Carlo and is stored in a TDR file named `30nm_end6.tdr`. Figure 16 shows the results of the kinetic Monte Carlo (KMC) simulation with the gate material and gate oxide removed. The light blue dots represent arsenic point defects, and the dark blue dots represent boron point defects.

Figure 16 Discrete dopant positions generated by Sentaurus Process Kinetic Monte Carlo



The command file `nmos.cmd` presented here is used to generate the mesh and doping for the structure.

Note:

Before running `nmos.cmd`, a 3D boundary file named `nmos_bnd.tdr` must exist that is consistent with the KMC particle file. If the final generated structure is to be used in Sentaurus Device, the boundary file must also contain the proper electrodes needed for device simulation.

The `Definitions` section of the command file defines the profiles and refinements that will later be used (in the `Placements` section) to create the mesh and doping for the structure:

```
Title "Specifying doping with particle profiles"
Definitions {
   Constant "PolyGateDoping" {
      Species = "ArsenicActiveConcentration"
      Value = 8e+19
   }
   Particle "BoronParticles" {
      ParticleFile = "30nm_end6.tdr"
      Species = "BoronActiveConcentration"
      ScreeningFactor = 2.5e6
      AutoScreeningFactor
      Normalization
   }
   Particle "ArsenicParticles" {
      ParticleFile = "30nm_end6.tdr"
      Species = "ArsenicActiveConcentration"
      ScreeningFactor = 1.0e7
      AutoScreeningFactor
      Normalization
   }
   Refinement "GlobalRefinement" {
      MaxElementSize = ( .020 .020 .020 )
      MinElementSize = ( .002 .002 .002 )
```

```
      RefineFunction = MaxTransDiff(Variable = "DopingConcentration",
                                    Value = 1)
   }
   Refinement "InterfaceRefinement" {
      MaxElementSize = ( .008 .004 .0002 )
      MinElementSize = ( .002 .002 .0001 )
      RefineFunction = MaxTransDiff(Variable = "SpatialCoordinates",
                                    Value = 1e-10)
   }
}
```

The `Constant` definition defines the doping that will be used for the polysilicon gate. Two `Particle` profiles are used to obtain separately the boron and arsenic discrete dopants from the KMC particle file. Separate `ScreeningFactor` values are specified for these two `Species`. Specifying `AutoScreeningFactor` generally results in a smoother and more accurate final profile in structures where there are large changes in dopant density. The `Normalization` option compensates for doping loss of dopants located near the boundary. The command file also includes a global refinement definition based on doping and an interface refinement definition based on spatial coordinates. The latter definition is intended to force a grid refinement using the specified element sizes.

The `Placements` section of the command file specifies how the profile and refinement definitions are used to create the structure:

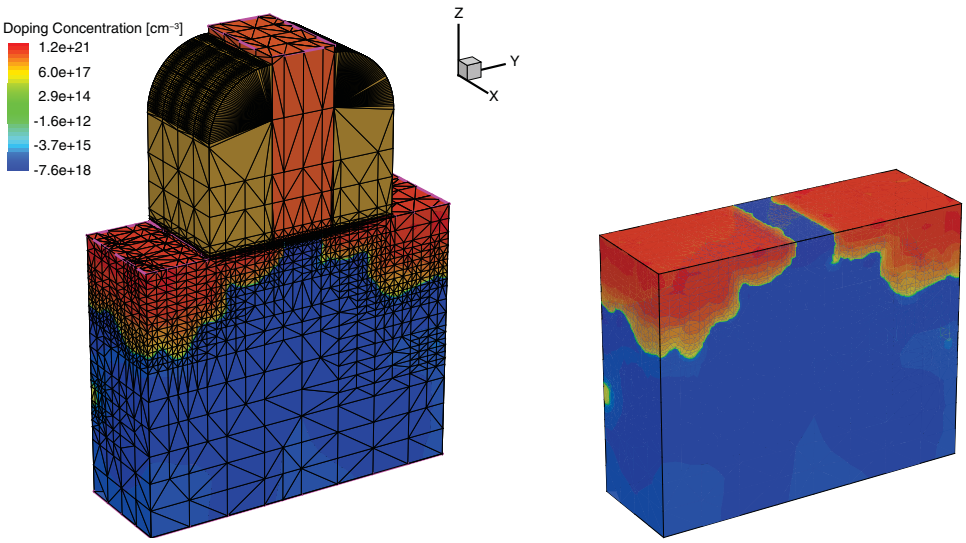
```
Placements {
   Constant "PolyGateDopingPlacement" {
      Reference = "PolyGateDoping"
      EvaluateWindow { Element = material ["PolySilicon"] }
   }
   Particle "ArsenicParticlesPlacement" {
      Reference = "ArsenicParticles"
      EvaluateWindow { Element=material ["Silicon"] }
   }
   Particle "BoronParticlesPlacement" {
      Reference = "BoronParticles"
      EvaluateWindow { Element=material ["Silicon"] }
   }
   Refinement "GlobalRefinementPlacement" {
      Reference = "GlobalRefinement"
      RefineWindow = material ["Silicon"]
   }
   Refinement "InterfaceRefinementPlacement" {
      Reference = "InterfaceRefinement"
      RefineWindow = Cuboid [(0.00 0.060 0.0000) (0.05 0.090 -0.0008)]
   }
}
```

The gate material (polysilicon) is uniformly doped using the `Constant` profile given in the `Definitions` section. The `Particle` profiles are placed only in the silicon portion of the structure. The global refinement (based on doping) is also only performed in silicon. The

interface refinement is confined to the channel region of the structure and to within 8 Å of the interface.

Executing this command file generates a TDR file named `nmos_msh.tdr`. The generated structure is shown in Figure 17.

Figure 17 Mesh and doping for the structure generated by Sentaurus Mesh



Generating 2D Mesh With Continuous Doping Obtained From 3D KMC File Containing Particle Information

This example illustrates the use of the feature that generates a continuous profile on a 2D mesh from a 3D KMC file containing particle information. This feature allows you to evaluate the 3D doping profile and to transfer those onto a 2D mesh.

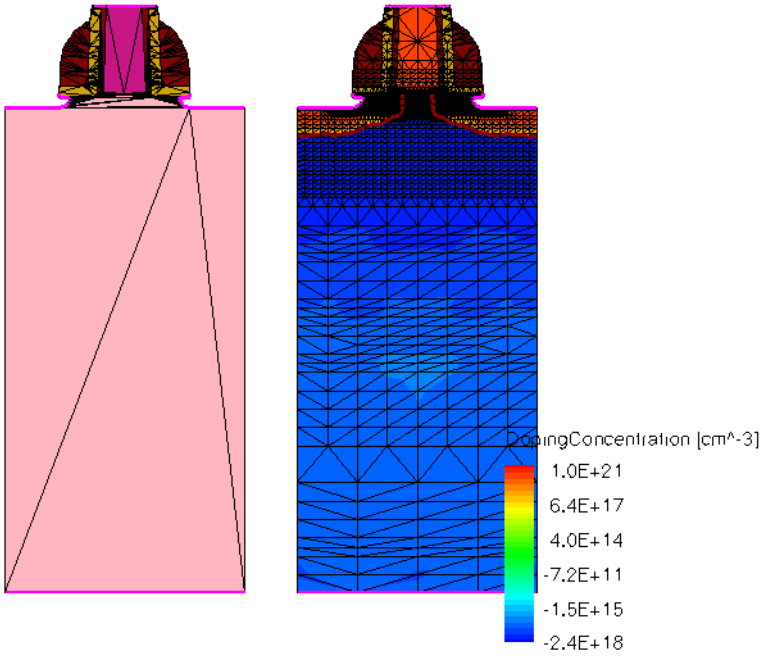
Here are the `Particle` subsections of the `Placements` section of the command file:

```
Placements {
   Particle "BoronParticles" {
      ParticleFile = "n26_final.tdr"
      Species = "BoronActiveConcentration"
      ScreeningFactor = 3.5e6
      AutoScreeningFactor
      Normalization
      BoundaryExtension = 0.05
      Divisions = 10
   }
   Particle "ArsenicParticles" {
      ParticleFile = "n26_final.tdr"
      Species = "ArsenicActiveConcentration"
```

```
        ScreeningFactor = 1.1e7
        AutoScreeningFactor
        Normalization
        BoundaryExtension = 0.05
        Divisions = 10
    }
}
```

The parameter boundary extension is used internally as the thickness of a 3D structure generated by extruding a 2D structure in the z-direction. The KMC file containing the particle information is mapped onto this 3D structure. For each 2D mesh point, a number of points equal to the number of divisions is created, each separated by an equal amount in the z-direction, and doping is computed on these points. The average doping is computed and is assigned to a 2D mesh point. Figure 18 shows the input boundary and generated mesh with particle profile.

Figure 18 Two-dimensional boundary and mesh with doping obtained from a 3D KMC file



Performing Interface Refinement

Interface refinement is specified in a similar way to the refinement on analytic functions. To perform interface refinement, define a `RefineFunction` of type `MaxLengthInterface` and specify the pair of materials defining the interface, the initial thickness, and a factor used to define how this thickness should grow into the material.

Examples

Refine silicon at the oxide interface, starting with a layer of 0.02 μm and gradually increasing the thickness by 1.4 times:

```
RefineFunction = MaxLenInt(Interface("Silicon","Oxide"), Value = 0.02,
                           Factor = 1.4)
```

Refine all interfaces, creating a single layer of 0.01 μm:

```
RefineFunction = MaxLengthInterface(Interface("All","All"), Value=0.01)
```

Refine all contacts in the mesh:

```
RefineFunction = MaxLenInt(Interface("All","Contact"), Value=0.01)
```

Refine around a single contact:

```
RefineFunction = MaxLengthInterface(Interface("All","Gate"),
                                    Value=0.01, UseRegionNames)
```

By default, interface refinement is performed only on the first material of the specified pair of materials describing the interface. To refine on both sides of the interface, use the `DoubleSide` option:

```
RefineFunction = MaxLenInt(Interface("Silicon","Contact"),
                           Value=0.01, DoubleSide)
```

Ignoring Interfaces Between Regions of the Same Material

When Sentaurus Mesh performs refinement across interfaces, it internally splits the edges crossing the interfaces into segments that are contained completely inside each region. Then, it proceeds to analyze the refinement criteria on each segment independently. In some applications, the doping concentration on each region is constant, so no refinement is applied since the gradient along each segment is zero.

However, sometimes, you might want to define different doping concentrations on adjacent regions of the same material and might want the code to ignore the interface between those regions so that Sentaurus Mesh can refine across the interface. In this case, specify the option `skipSameMaterialInterfaces` in the `AxisAligned` section of the command file to obtain the required effect.

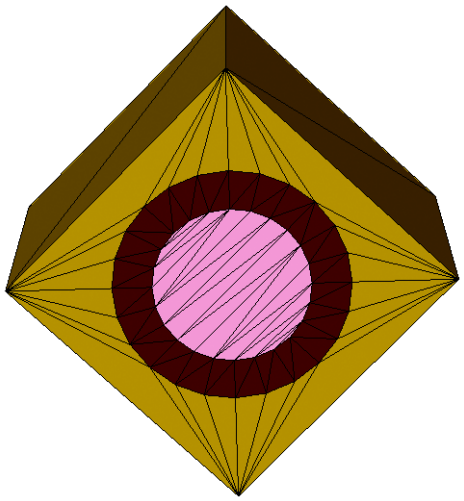
Offsetting Mesh Generation

This section presents examples illustrating the use of the offsetting mesh generator.

Simple Example

This example shows all the relevant parameters in the `Offsetting` section that Sentaurus Mesh supports.

Figure 19 Input structure for simple example



Command File

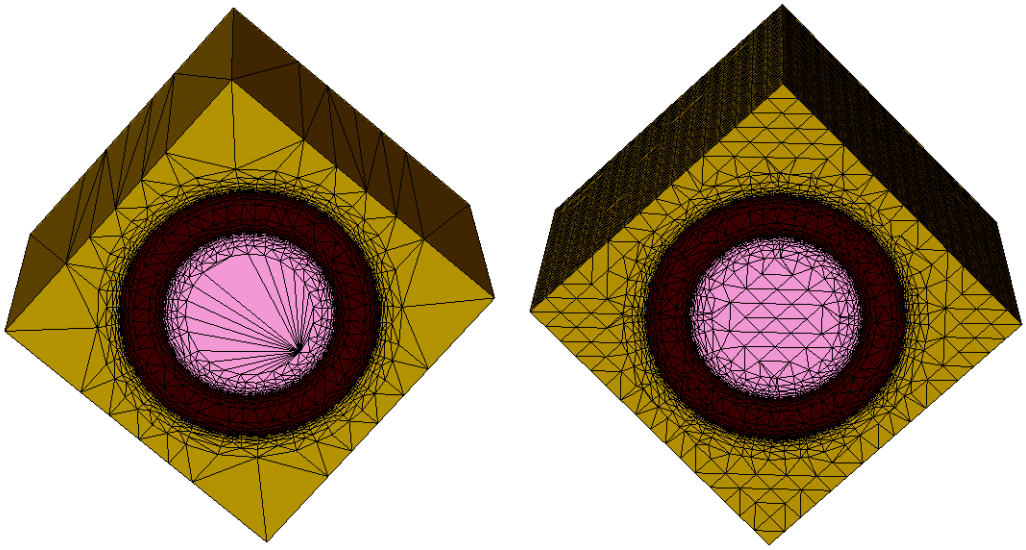
```
Title ""
IOControls {
   EnableSections
}

Definitions {
   Refinement "R5" {
      MaxElementSize = ( 0.026 0.026 0.026 )
   }
}

Placements {
   Refinement "GDJ_RP" {
      Reference = "R5"
      RefineWindow = Cuboid [(-0.2 -0.2 0) (0.20 0.2 0.5)]
   }
}
```

```
Offsetting {
   noffset {
      hlocal=0
   }
   noffset material "Silicon" {
      maxlevel = 5
   }
   noffset material "Oxide" {
      maxlevel = 5
   }
   noffset material "Silicon" "Oxide" {
      hlocal=0.002
      factor=1.5
   }
   noffset material "Oxide" "Silicon" {
      hlocal=0.002
      factor=1.5
   }
   noffset region "RTrench" "RBulk" {
      hlocal=0.002
      factor=1.5
   }
   noffset region "RBulk" "RTrench" {
      hlocal=0.002
      factor=1.5
   }
}
```

In this command file, the global `hlocal` value is set to zero. Later, a nonzero `hlocal` value and `factor` are set using the material interface section. The default `maxlevel` value of 200 is also reset to 5 using a material section.

If the command file only includes the `Offsetting` section, without specifying any refinement criteria in the `Definitions` and `Placements` sections, the resultant mesh will be coarse as shown in Figure 20 on page 119 (*left*). With the refinement specified in the command file, the generated mesh is shown in Figure 20 (*right*).

Figure 20 *(Left) Coarse mesh generated with layers and without refinement criteria, and (right) mesh generated with layers along with refinement criteria*



Layering From All Boundaries

```
Title ""
IOControls {
   EnableSections
}

Definitions {
   Refinement "RefinementDefinition_1" {
      MaxElementSize = ( 0.2 0.2 0.2 )
      MinElementSize = ( 0.001 0.001 0.001 )
   }
}

Placements {
   Refinement "RefinementPlacement_1" {
      Reference = "RefinementDefinition_1"
      RefineWindow = region ["R_Silicon"]
   }
   Refinement "RefinementPlacement_2" {
      Reference = "RefinementDefinition_1"
      RefineWindow = region ["R_Oxide"]
   }
}

Offsetting {
   noffset {
      hlocal=0.01
      maxlevel=6
```

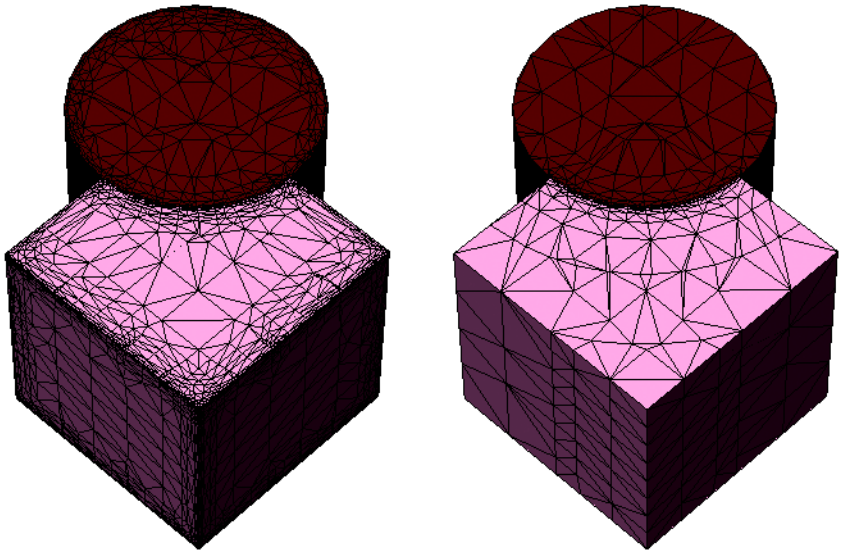
```
   }
   noffset material "Oxide" "Silicon" {
      factor=2
   }
}
```

In this command file, the global parameter `maxlevel` is set to 6 and the global `hlocal` is a nonzero value. This results in layering only from interfaces excluding exterior boundaries. If layering is required from all interfaces including boundaries that do not share any, the `Offsetting` section can be modified as shown:

```
Offsetting {
   noffset {
      maxlevel=6
   }
   noffset material "All" "All" {
      hlocal=0.01
      factor=2
   }
}
```

The string `"All"` refers to all materials in the input structure. Figure 21 shows the resultant meshes that are generated using the original and modified `Offsetting` sections.

Figure 21 (Left) Mesh with layering from all interfaces and from boundaries with nonzero global hlocal value and (right) mesh generated with modified Offsetting section having layering only at interfaces



Localizing the Refinement Using Cuts

The axis-aligned mesh generator always creates the mesh by refining an initial box, which contains the entire device. This creates problems when the external shape of the device must be modified, because this will change the bounding box of the device, thereby altering the location of the mesh nodes.

To help resolve this situation, you can define an initial array of boxes from which to start the refinement, by using the `xCuts`, `yCuts`, and `zCuts` parameters in the `AxisAligned` section.

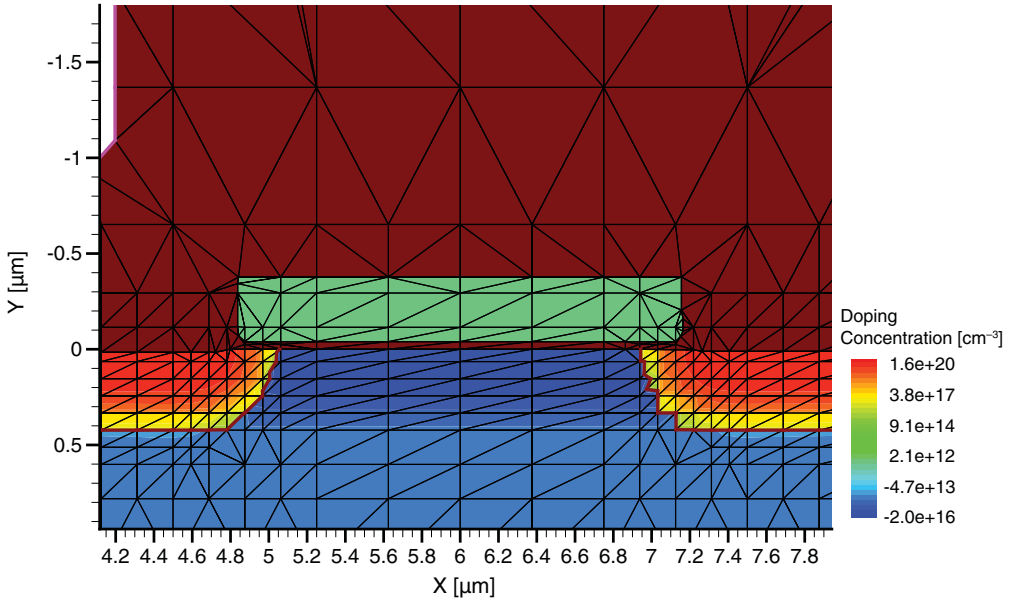
Each cut defines and initial refinement line that runs throughout the entire device. The boxes created with these initial lines can be refined independently of each other. Therefore, if the shape of the device changes and the cuts are adjusted accordingly, the mesh should stay the same in the sections where the boxes have remained unchanged.

The following example uses two lines to generate a total of three initial boxes. The lines are placed on either side of the channel and can be used to parameterize a set of structures where the only difference is the channel length:

```
AxisAligned { xCuts = (4.837 7.156) }
```

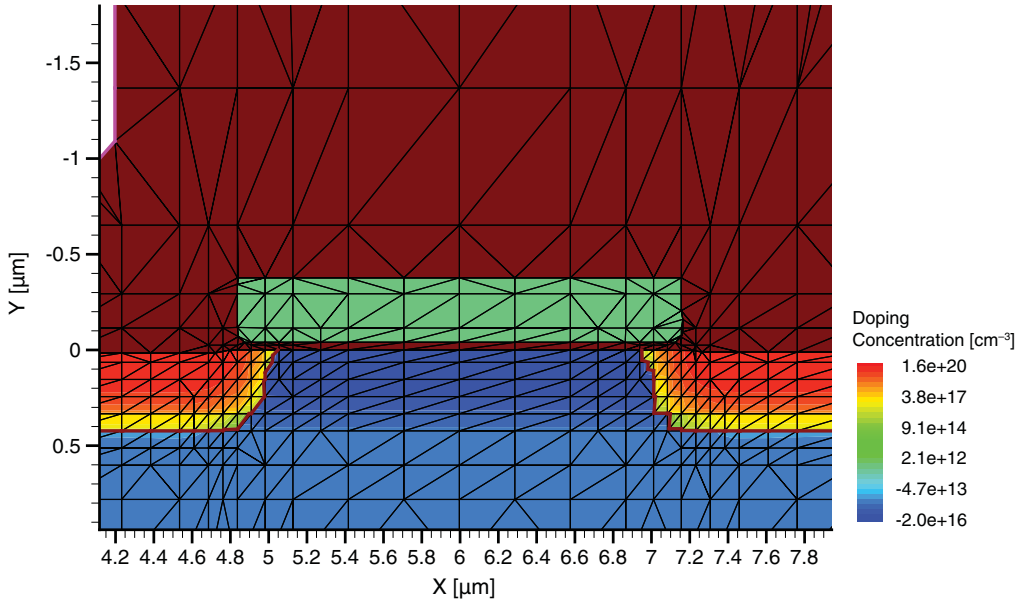
Figure 22 displays the channel of an NMOS structure that has been refined using standard refinement parameters.

Figure 22 NMOS structure refined using standard refinement



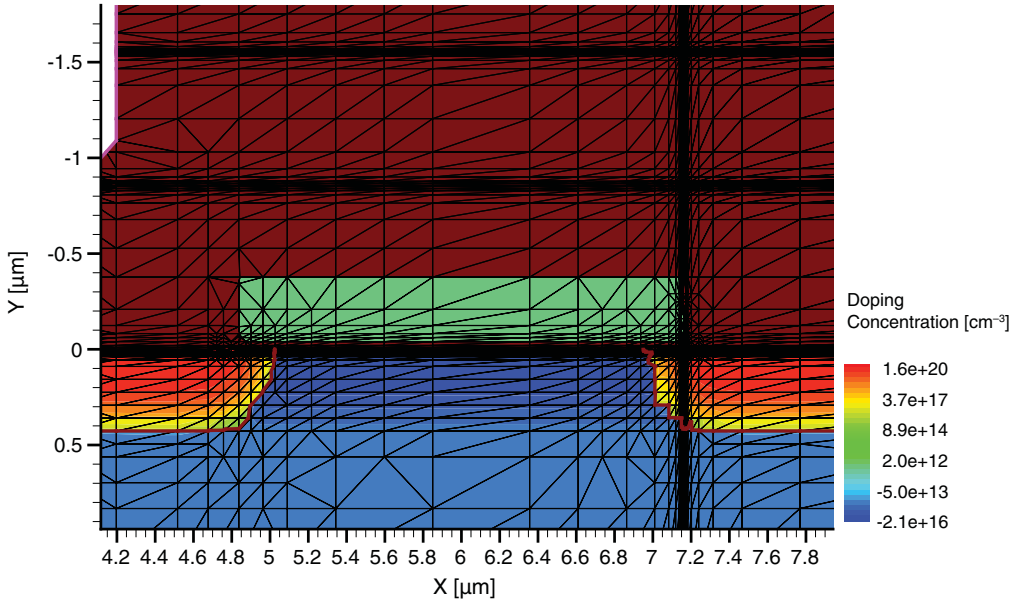
When the `xCuts` parameter is used, two refinement lines are placed at locations x=4.837 and x=7.156 (see Figure 23). If the channel length is increased by 0.01 µm, the second x-line could be placed at 7.157, thereby preserving the mesh on either side of the channel.

Figure 23 NMOS structure refined using the xCuts parameter



Another possibility is to use the `fitInterfaces` parameter in the `AxisAligned` section of the command file. This parameter works best on simple devices where all interfaces are axis aligned. Figure 24 shows that the device can be overrefined when the interfaces are not simple.

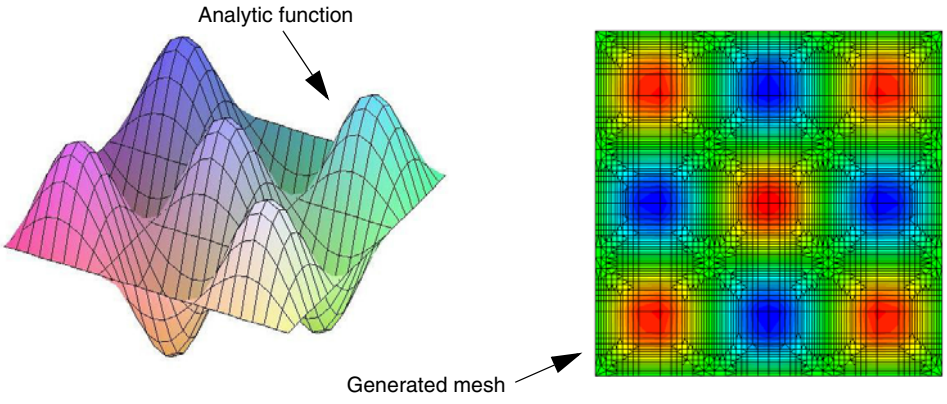
Figure 24 NMOS structure refined using the fitInterfaces parameter



Using Analytic Functions for Refinement I

Figure 25 illustrates the use of general analytic functions to specify profiles.

Figure 25 Use of analytic refinement functions



The function $0.1\sin(x)\sin(y)$ is used as a profile and linear interpolation (`"ElectrostaticPotential"`) is used to compute the required local element size.

The following command file excerpt illustrates the syntax:

```
Definitions {
   Refinement "Region_1" {
      MaxElementSize = (1 1)
      MinElementSize = (0.01 0.01)
      RefineFunction = MaxTransDiff(Variable = "ElectrostaticPotential",
                          Value = 0.01)
   }
   AnalyticalProfile "Profile_1" {
      Species = "ElectrostaticPotential"
      Function = General(init="a=0.1",function = "a*sin(x)*sin(y)",
                         value = 0)
   }
}

Placements {
   Refinement "Region_1" {
      Reference = "Region_1"
   }
   AnalyticalProfile "Profile_1" {
      Reference = "Profile_1"
      EvaluateWindow {
         Element = rectangle [(0 0), (9.43 9.43)]
      }
   }
}
```

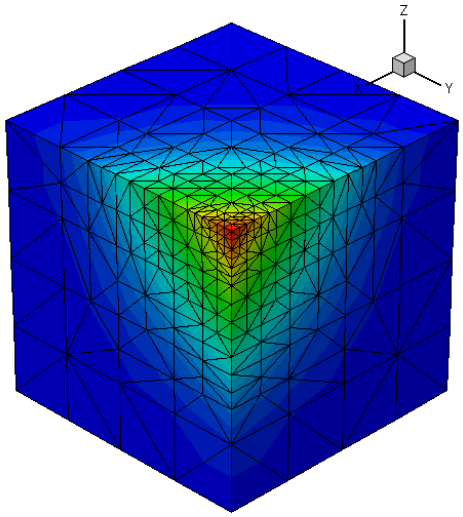
Using Analytic Functions for Refinement II

This example illustrates the use of a general analytic function to prescribe 3D refinement based on a 3D analytic function. The domain is a cube. Figure 26 on page 125 shows the generated mesh.

```
Definitions {
   Refinement "Region_1" {
      MaxElementSize = (4 4 4)
      MinElementSize = (0.01 0.01 0.01)
      RefineFunction = MaxTransDiff(Variable = "ElectrostaticPotential",
                          Value = 10000.0)
   }
   AnalyticalProfile "Profile_1" {
      Species = "ElectrostaticPotential"
      Function = General(init="a=0.1",function = "a*x*x*y*y*z*z",
                         value = 0)
   }
}

Placements {
   Refinement "Region_1" {
      Reference = "Region_1"
```

```
        }
        AnalyticalProfile "Profile_1" {
           Reference = "Profile_1"
           EvaluateWindow {
              Element = cuboid [ ( 0 0 0 ), ( 10 10 10 )]
           }
        }
    }
```

Figure 26 Use of analytic refinement functions



4

Tensor-Product Examples

This chapter presents examples that demonstrate the applications of the Tensor section of the command file.

Simple Cube

This example illustrates the use of parameters such as `maxBndCellSize`, `maxCellSize`, refinement using `window`, and `grading`. The following command file generates a tensor-product mesh:

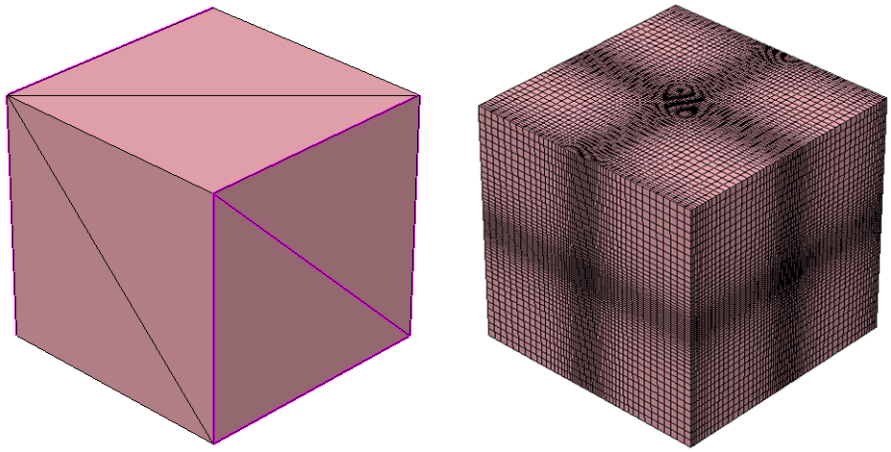
```
Tensor {
   Mesh {
      maxBndCellSize direction "x" 0.001
      maxCellSize region "Region_0" 0.1
      window "testbox" 0.8 1.2 0.8 1.2 0.8 1.2
      minNumberOfCells window "testbox" 20
      grading = { 1.1 1.1 1.1 }
   }
}
```

Figure 27 shows the geometry of this example. In this command file, the parameter `maxBndCellSize` is constrained in the x-direction by specifying the `direction` option. As a result, clustering is obtained only near the boundaries that are normal to the x-axis (see Figure 27 (*right*)).

The parameter `maxCellSize` is specified within a `region`. Since no `direction` option is used with this parameter, the mesh generator tries to obtain the same cell size in all three directions. To obtain refinement in the center of the geometry, the parameter `window` is defined. The required refinement within a window can be obtained by specifying either `maxCellSize` or `minNumberOfCells`.

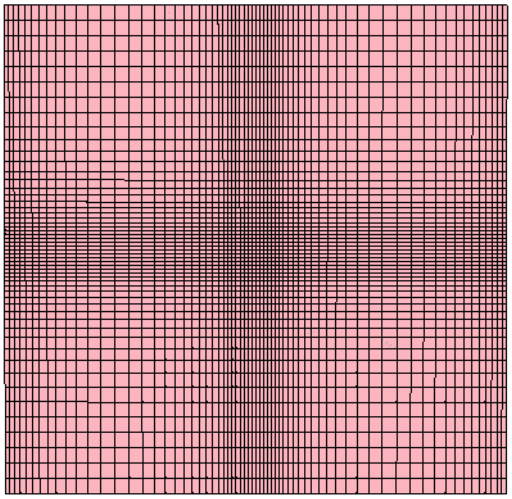
In this example, `minNumberOfCells` specifies the refinement. As a result, clustering of lines is visible in Figure 27 (*right*). The `grading` parameter is used to obtain a smooth variation of the cell sizes between various cell sizes (see Figure 28 on page 127).

Figure 27 Geometry of a cube with a width of 2.0 units in each direction: (left) the input geometry and (right) corresponding tensor mesh



In Figure 27, the clustering of cells normal to the x-axis is visible as per the specification of the minimum boundary cell size parameter in the command file. The refinement in the center of the cube is due to the specification of the `minNumberOfCells` parameter. This refinement is constrained to a `"testbox"` window.

Figure 28 Smooth variation of cell size from minimum to maximum cell size according to the specified grading factor



Using Boundary and Command Files to Generate Doping and Refinement

This example shows how to use the tensor-product mesh generator to represent an approximation of the actual regions defined in a boundary file. The geometry is shown in Figure 29. In this case, the default mesh generation parameters are used. A doping refinement section is provided in the command file.

Figure 29 Geometry of an input file

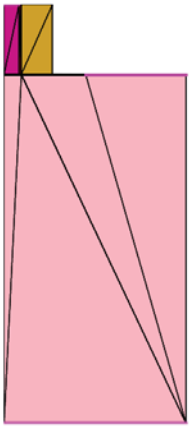


Figure 30 shows the tensor mesh and the corresponding doping data. Note that the axis-aligned interfaces are represented accurately by the tensor mesh. However, if the geometry has a curved region, the resulting mesh will be an approximation of the boundary, as shown in Figure 31.

Figure 30 (Left) Tensor mesh and (right) the doping data interpolated onto tensor mesh

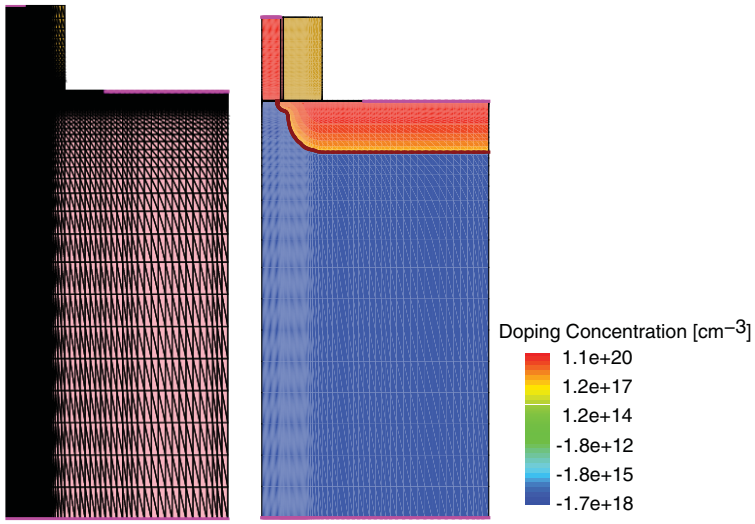
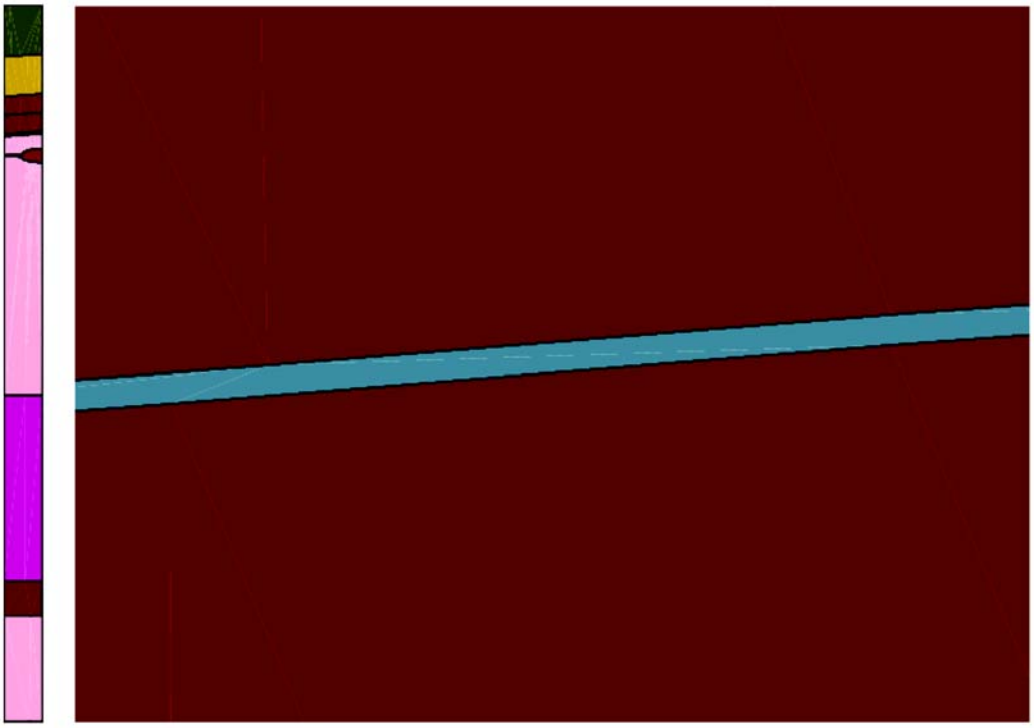


Figure 31 (Left) Actual curve region in the geometry and (right) corresponding approximation of this region in tensor mesh

Thin Regions

This example shows how insufficient cell resolution in a region results in elements that are one dimension less than the model dimension. The geometry contains a thin aluminum region shown in Figure 32. Since the local cell size is not small enough to resolve this aluminum region, this region is represented as 1D elements in the output as shown in Figure 33.

Figure 32 (Left) Geometry of the example and right) detail of thin aluminum region

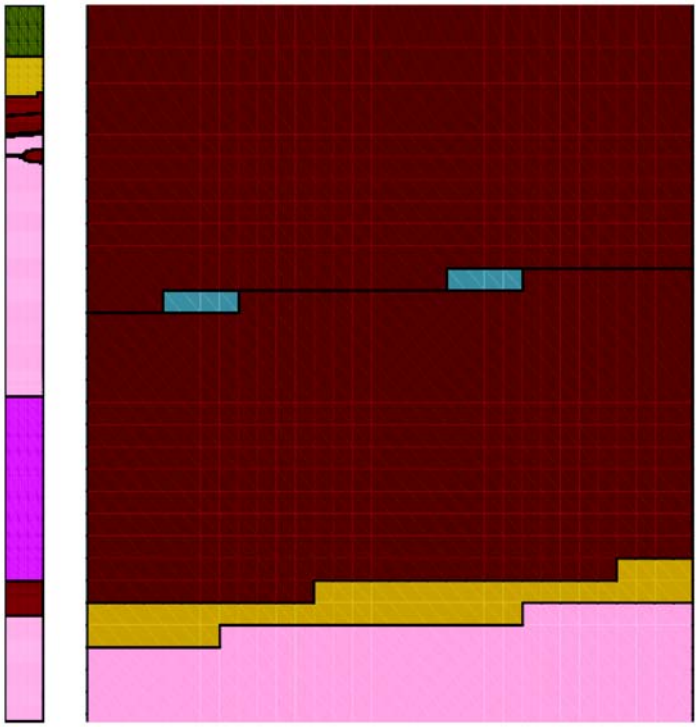


In Figure 33, the aluminum region is not resolved properly as the locally cell size is larger than the aluminum region. As a result, this region contains two faces and the rest of the region is represented as a set of edges connecting the two faces. Similarly, in 3D models, unresolved regions and contacts will be written as thin surface sheets. Sentaurus Mesh issues a warning indicating the presence of completely or partially unresolved regions.

The following is the command file used to generate the tensor-product mesh:

```
Tensor {
   Mesh {
      maxCellSize material "Oxide" 0.08
      maxCellSize material "Silicon" 0.08
      grading = { 1.1 1.1 1.1 }
   }
}
```

Figure 33 (Left) Corresponding tensor mesh and (right) detail of tensor mesh near aluminum region



Computing Cell Size Automatically (EMW Applications)

This example demonstrates how to compute cell sizes automatically using the tensor-product mesh generator, which reads first the `Mesh` subsection and then the `EMW` subsection of the `Tensor` section. The cell sizes for the materials are computed using the optical database table, which is defined in either the user-defined Sentaurus Device parameter file or the default Sentaurus Device parameter file.

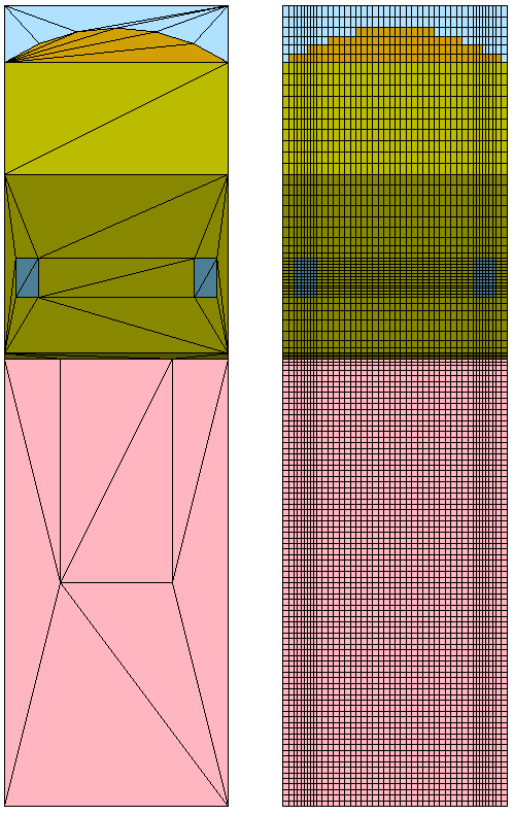
The computed cell size is a function of the:

- Wavelength
- Nodes per wavelength
- Norm of a complex refractive index

In the following example, the wavelength equals 0.6 µm and the number of nodes per wavelength is 15. In addition, a maximum number of nodes per wavelength is specified (20) to compute a minimum cell size, which controls the largest stable time step for EMW. Figure 34 shows the generated tensor mesh.

```
Tensor {
   EMW {
      CRI WavelengthDep real imag
      wavelength = 0.6
      npw = 15
      maxnpw = 20
   }
}
```

Figure 34 (Left) Boundary and (right) corresponding tensor mesh generated by using the EnableEMW option in the IOControls section of the command file



5

Tools Section

This chapter illustrates the use of the Tools section of the command file for Sentaurus Mesh.

Activating the Tools Section

To activate the `Tools` section in the command file of Sentaurus Mesh, specify either `EnableSections` or `EnableTools` in the `IOControls` section of the command file (see IOControls Section on page 16).

Reflecting and Extruding the Mesh

In the following example, a 2D structure is taken and reflected, and then a 3D mesh is generated by extruding the reflected mesh:

```
Tools {
   Reflection {
      axis = xmin
      map "R.PolyReox" = "R.PolyReox_new"
   }
   Extrude {
      extension = 0.1
      steps = 5
   }
}
```

Figure 35 shows the input mesh. The first step reflects the mesh structure about the `xmin` coordinate. The `map` statement renames the mirrored region `R.PolyReox` to `R.PolyReox_new`.

Figure 35 Input structure

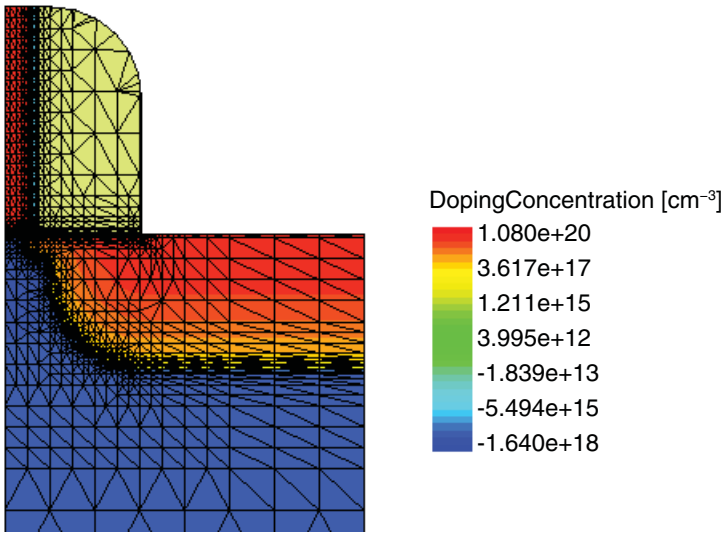
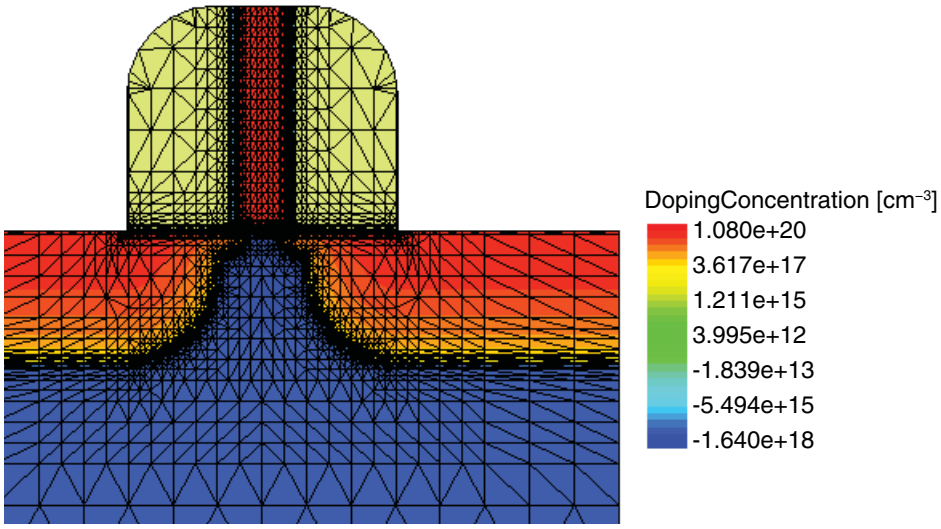


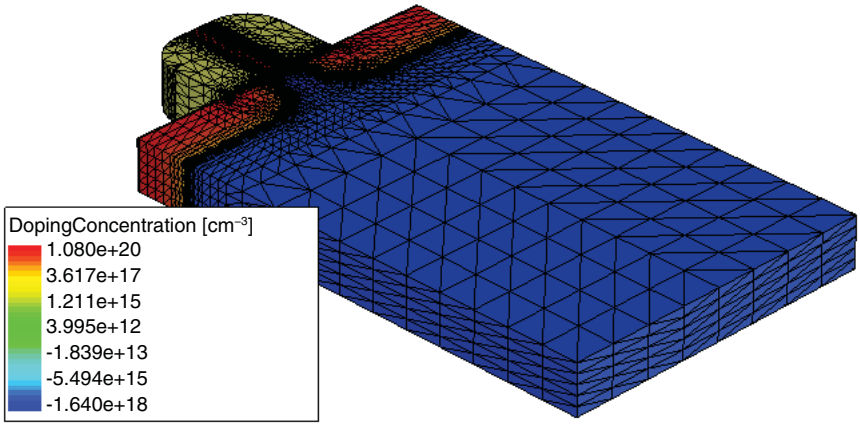
Figure 36 shows the reflected mesh. Since `Extrude` is specified in the command file, the reflected mesh becomes input to this tool.

Figure 36 Reflected mesh



A 3D mesh is generated by extruding the reflected mesh in the z-direction by 0.1 µm. Then, this mesh is divided into five sections along the z-direction as shown in Figure 37.

Figure 37 Mesh after being extruded in the z-direction



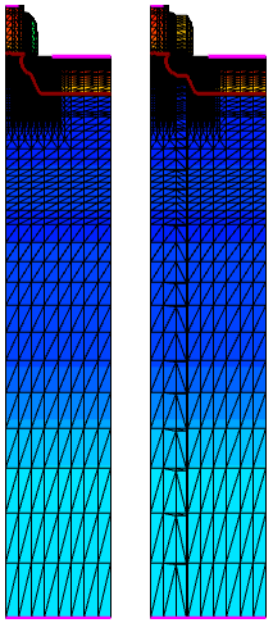
Stretching a Mesh

This example demonstrates how to stretch a mesh. The command file contains information about the location of the starting point of the stretch, the direction of the stretch, and the length of the stretch:

```
Tools {
   Stretch {
      location = (0.12 0.005 0)
      direction = X
      length = 0.05
   }
}
```

Figure 38 (*left*) shows the input mesh. With the information in the command file, a new column of elements is added at the specified location. The output stretch mesh is shown in Figure 38 (*right*). After stretching, the length of the mesh in the specified direction increases by a specified length. The unit of length is the same as the input mesh.

Figure 38 (Left) Input mesh before stretch and (right) mesh after stretch operation



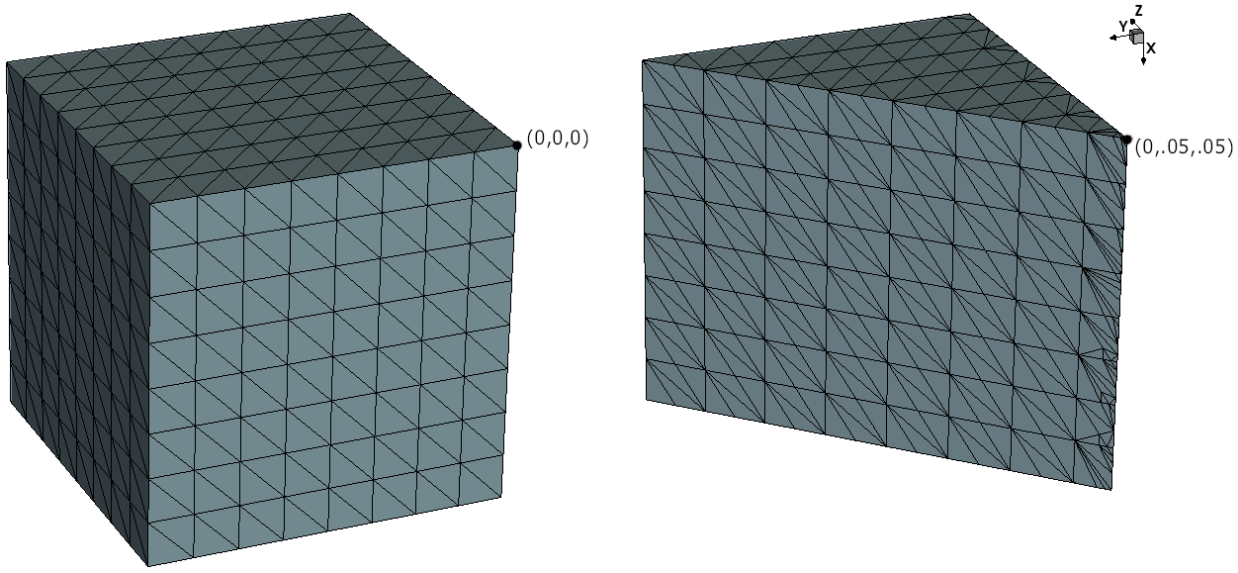
Cutting a 3D Mesh

In this example, a cube mesh is taken as input and three cutting planes are specified to create a wedge:

```
Tools {
   Cut {
      normal = (0 0 1)
      location = (0.05 0.05 0.05)
   }
   Cut {
      normal = (0 -0.70711 0.70711)
      location = (0.05 0.05 0.05)
   }
   Cut {
      normal = (0 1 0)
      location = (0.05 0.05 0.05)
   }
}
```

The input mesh is processed with the first cutting plane, and its output is given as an input to the next cutting plane. Figure 39 shows the input mesh and the final output mesh.

Figure 39 (Left) Input mesh and (right) final wedge created by using three cutting planes



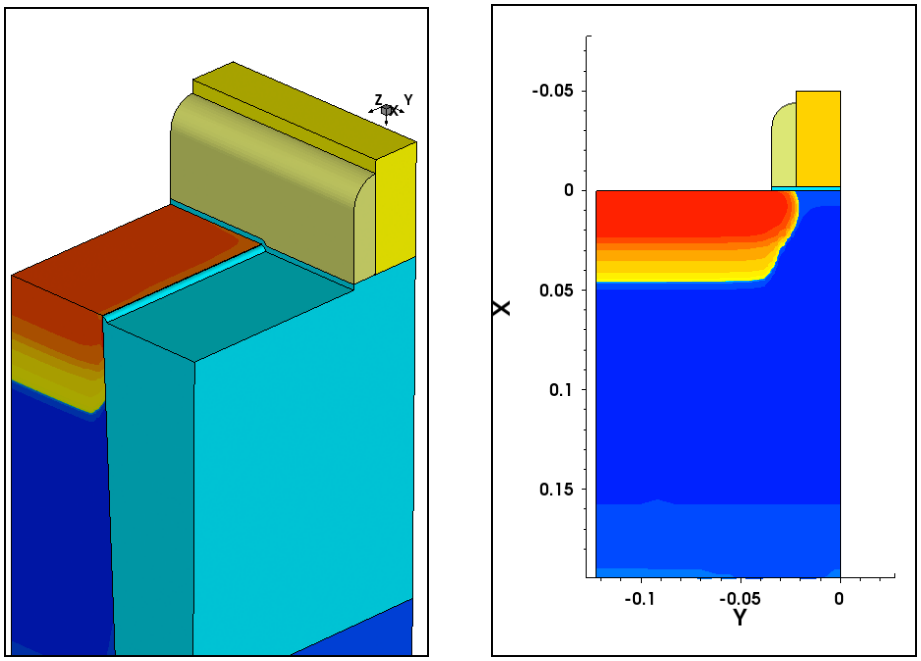
Slicing a 3D Mesh Using a Plane and Its Location

In the following example, a 3D mesh is sliced to obtain a 2D mesh:

```
Tools {
   Slice {
      normal = (0 1 0)
      location = (0 0.0075 0)
   }
}
```

Figure 40 (*left*) shows the input mesh, which is sliced with the y-plane placed at the specified location (`0 0.0075 0`). This results in a 2D mesh slice shown in Figure 40 (*right*).

Figure 40 (Left) Input 3D mesh and (right) 2D mesh slice generated from the input mesh



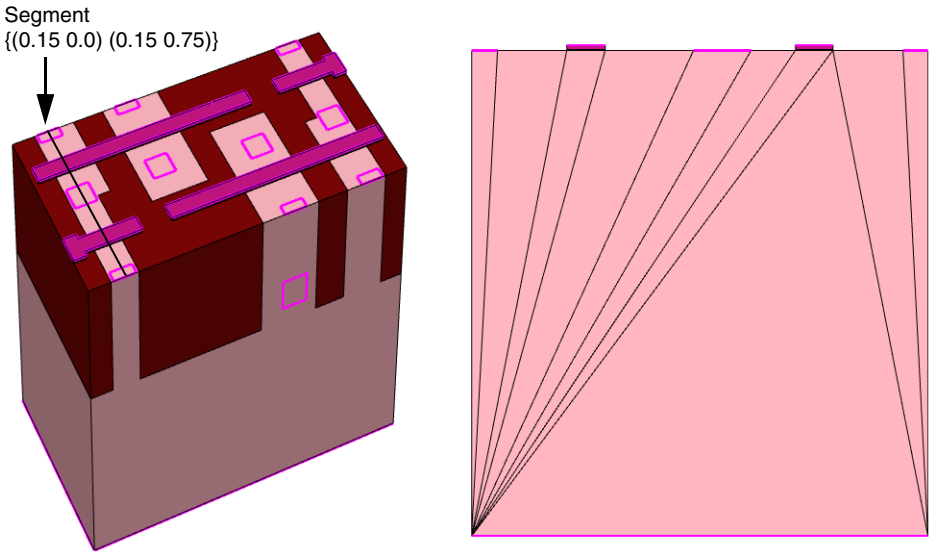
Slicing a 3D Mesh Using a Segment and a Direction

In this example, a 3D mesh is sliced to obtain a 2D mesh:

```
Tools {
   Slice {
      Direction = Z
      Startpoint = (0.15 0.0)
      Endpoint = (0.15 0.75)
   }
}
```

This `Tools` section specifies a segment with a starting point and an endpoint on a constant z-plane. With this information, a bounding box of the input structure (shown in Figure 41 (*left*)) is computed and used in the construction of a plane defined by the bounding box coordinates of `[(0.15 0.0 zmin) (0.15 0.75 zmax)]`. The input boundary is sliced with this plane and the result is shown in Figure 41 (*right*). This boundary slice file also contains transformation information that can be used to place this slice back into 3D space for later applications.

Figure 41 (Left) Input boundary and (right) generated 2D slice



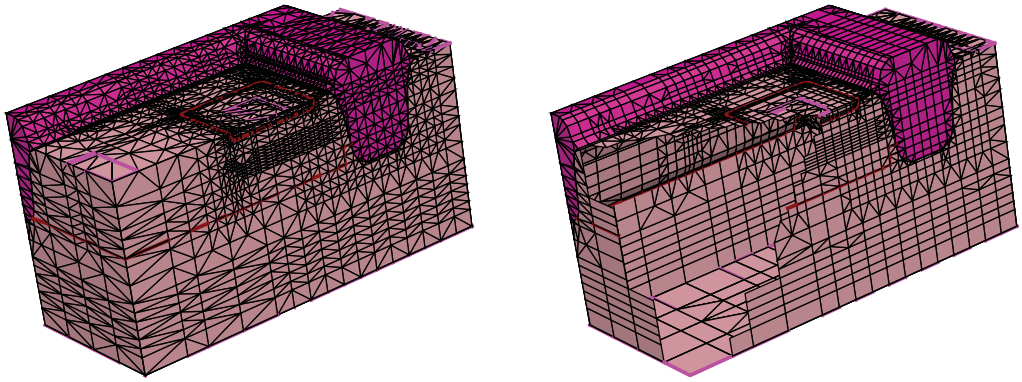
Converting a Tetrahedral Mesh to a Hybrid Mesh

The following example translates a mesh and converts it to a hybrid mesh:

```
Tools {
   Set Transformation {
      translation = ( 3 7 1 )
   }
   Apply Transformation
   Mesh2Hybrid
}
```

The log file contains information about a number of different element types in the converted mesh. In this example, approximately 55% of element reduction is achieved when compared to the input mesh.

Figure 42 (Left) Tetrahedral mesh and (right) corresponding hybrid element produced by the Mesh2Hybrid option



Generating Randomized Doping From Continuous Doping

The following example illustrates how to randomize a continuous doping. A mesh with continuous doping, along with a command file, is given as input:

```
Title "Doping Randomizer Example"
# Use "snmesh nmos00.cmd" to run. Assumes nmos00.tdr exists.
IOControls {
   EnableSections
}
Tools {
   RandomizeDoping {
      DopingAssignment = "Sano"
      # ContinuousContactDoping
      NumberOfRandomizedProfiles = 1
      FileIndex = 1
      Material "Silicon" {
         Species "BoronActiveConcentration" {
            ScreeningFactor = 2.5e6
            AutoScreeningFactor
         }
         Species "ArsenicActiveConcentration" {
            ScreeningFactor = 1.3e7
            AutoScreeningFactor
         }
      }
   }
}
```

The `"Sano"` method is used in this example. The number of randomized profiles is 1, and `FileIndex` is set to 1. Since only the material `Silicon` is specified, only silicon regions are randomized. Other materials retain their original continuous doping. Figure 43 shows the input structure mesh with continuous doping.

Figure 43 Input mesh with continuous doping

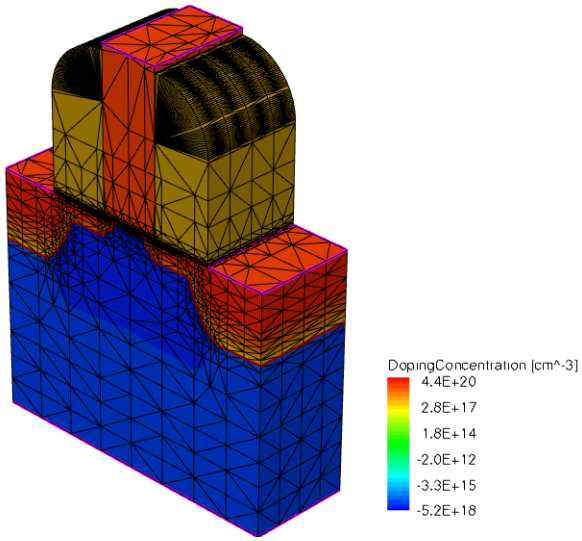
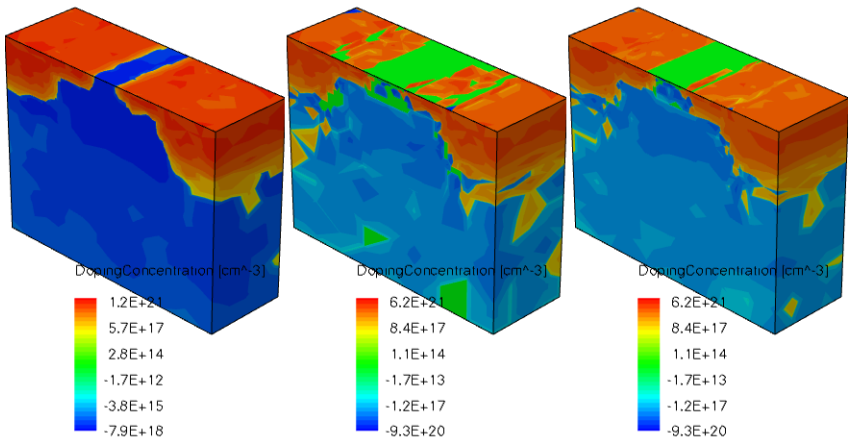


Figure 44 (*left*) shows the randomized doping generated by using `DopingAssignment= "Sano"`, and the command file is modified later to use the other methods.

Figure 44 Meshes with randomized doping generated by using the (left) Sano method, (middle) NGP method, and (right) CIC method



Creating Profiles in an Existing Mesh

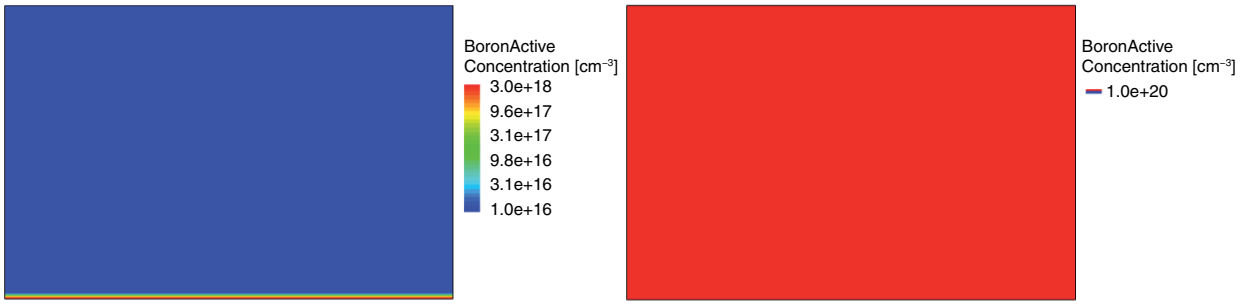
This example demonstrates how to create profiles. The command file contains information about an existing mesh and a mesh command file containing profile information. The profiles specified in the command file are created in the input mesh without changing the mesh itself.

The command file for this example is:

```
Tools {
   CreateProfiles {
      SrcMesh = "n6_0_msh.tdr"
      CmdFile = "n6_msh.cmd"
   }
}
```

The `CreateProfiles` subsection specifies the source mesh file (see Figure 45 (*left*)) and the mesh command file.

Figure 45 (Left) Input mesh showing BoronActiveConcentration profile before update and (right) mesh with updated BoronActiveConcentration



The mesh command file contains the following information related to the profile `BoronActiveConcentration`:

```
Title "Untitled"
Definitions {
   Constant "substrateDop" {
      Species = "BoronActiveConcentration"
      Value = 1e+20
   }
}
Placements {
   Constant "substrateDop" {
      Reference = "substrateDop"
      EvaluateWindow { Element = region ["substrate"] }
   }
}
```

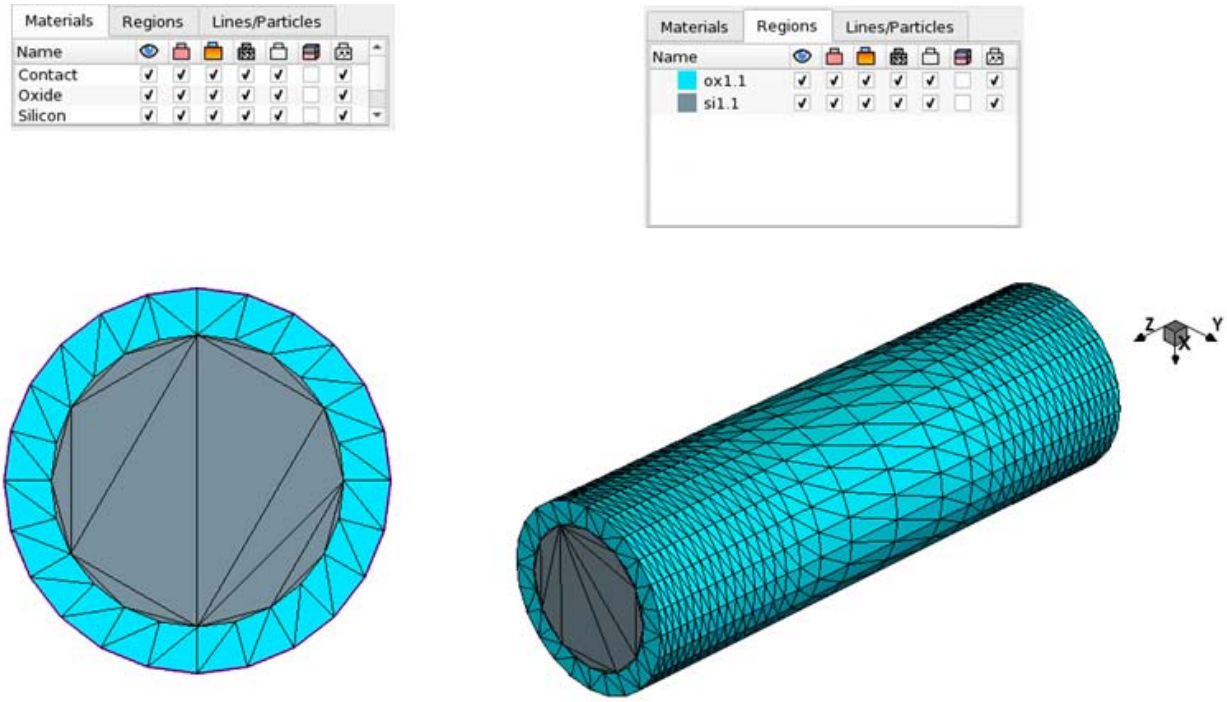
The `BoronActiveConcentration` profile in the input mesh is recreated without changing other profiles and, accordingly, the doping concentration is updated. The output mesh file is shown in Figure 45 (*right*).

Extruding and Remapping the Regions of a Mesh

In this example, you extrude a 2D structure as follows:

```
Tools {
   Extrude {
      spacingMethod = smooth
      zCuts = ((0.0 0.000275) (0.0055 0.5e-3) (0.0105 0.005)
               (0.0155 0.5e-3) (0.021 0.000275))
   }
}
```

Figure 46 (Left) Structure before extrusion and (right) structure after extrusion

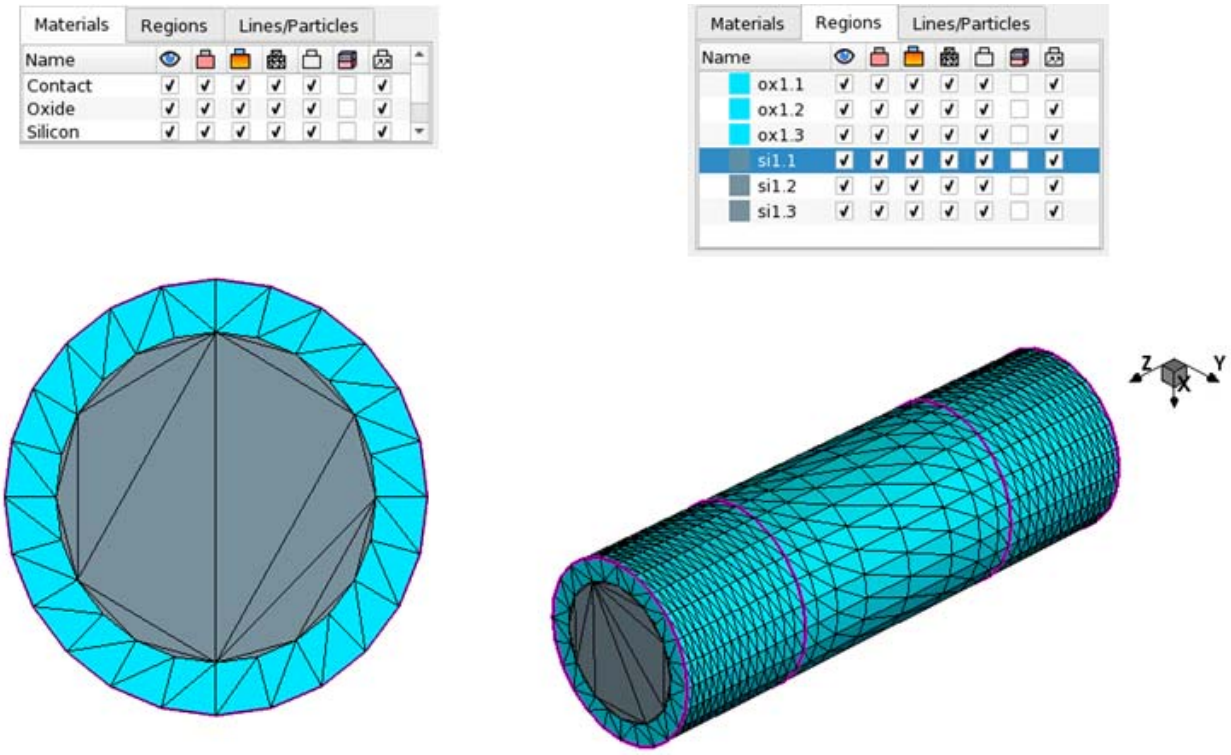


With the `zCuts` parameter, you can specify a `zCut` entry as a region boundary by setting the Boolean parameter in a `zCut` entry as `true` (see Extruding a Mesh on page 79). For example:

```
Tools {
   Extrude {
      spacingMethod = smooth
      zCuts = ((0.0 0.000275) (0.0055 0.5e-3 true) (0.0105 0.005)
               (0.0155 0.5e-3 true) (0.021 0.000275))
   }
}
```

This command splits the extruded mesh at region boundaries (z=0.0055 and z=0.0155) and creates subregions as shown in Figure 47.

Figure 47 (Left) Structure before extrusion and (right) structure after extrusion, which creates subregions (ox1.2 and ox1.3)



Changing Region Names

After extrusion, you can also change the region parameters such as its name and material (see Remapping Mesh Region Names and Region Materials on page 81). For example:

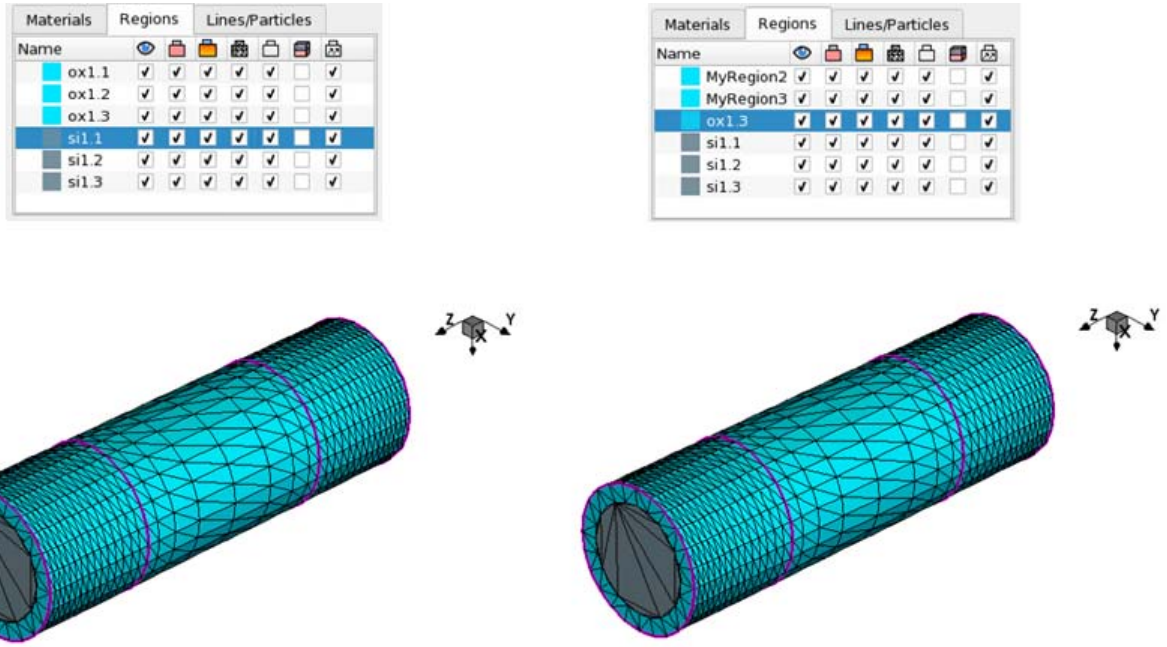
```
Tools {
   MapRegions {
      RegionChange = (("ox1.1" "MyRegion2") ("ox1.2" "MyRegion3")
                      ("ox1.7" "MyRegion4"))
   }
}
```

This section changes the names of valid regions as follows:

- `ox1.1` > `MyRegion2`
- `ox1.2` > `MyRegion3`

As `ox1.7` is not a valid region, the section has no effect.

Figure 48 (Left) Region names before change and (right) region names after change



Changing Region Materials

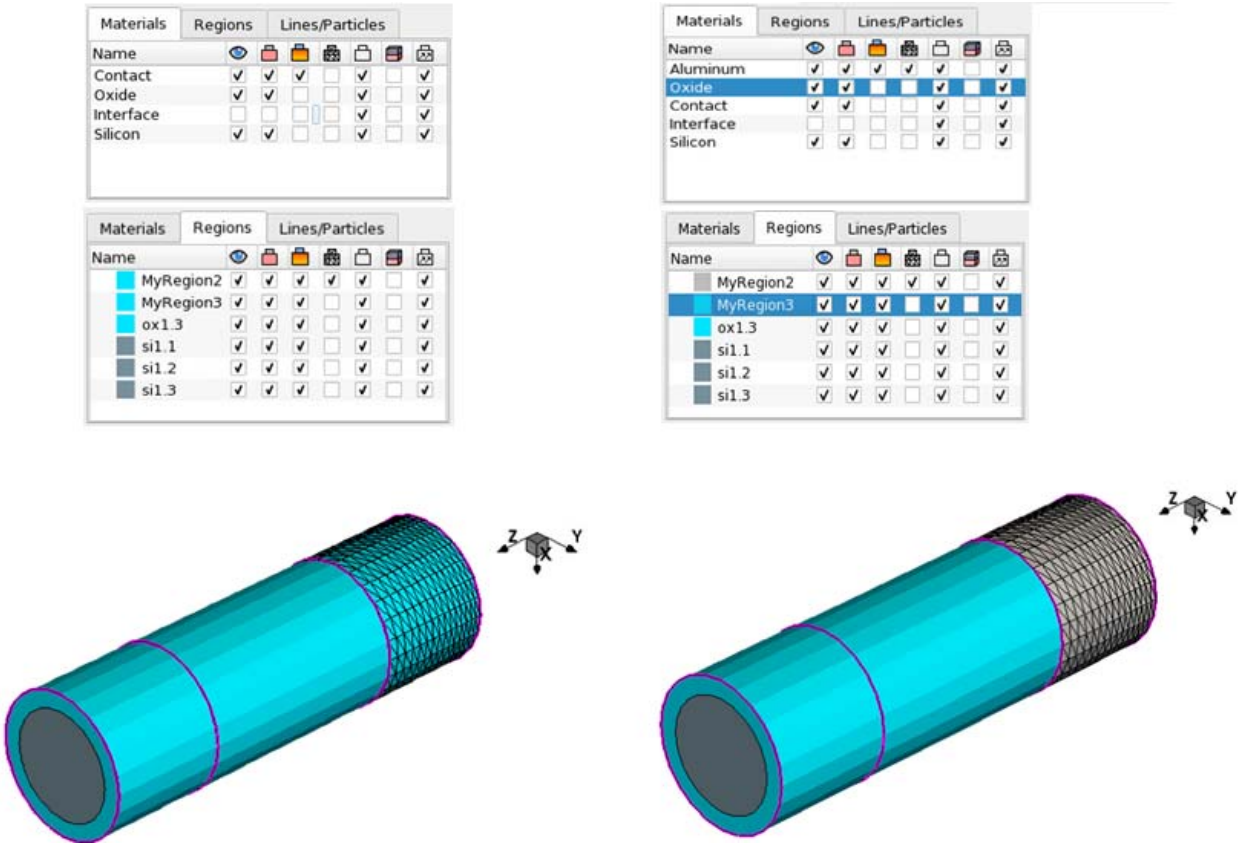
In addition to changing the name of a region, you can change its material (see Remapping Mesh Region Names and Region Materials on page 81). For example:

```
Tools {
   MapRegions {
      MaterialChange = (("MyRegion2" "Aluminum")
                        ("MyRegion3" "NonDatexMaterial")
                        ("UnnamedRegion" "SiliconCarbide"))
   }
}
```

The following changes occur when this section is executed:

- The material of `MyRegion2` changes from silicon to aluminum.
- As `NonDatexMaterial` is not a valid DATEX material, the material of `MyRegion3` is unchanged.
- As `UnnamedRegion` is not a valid region, the section has no effect.

Figure 49 (Left) Region materials before change and (right) region materials after change



6

Delaunization Algorithm

This chapter describes the delaunization algorithm used by Sentaurus Mesh.

Functionality of Delaunization Algorithm

A delaunization algorithm is available for 3D models. It is based on a conforming Delaunay triangulation–type of algorithm, but it is more stable, generating meshes for complex structures with sharp input angles that could not previously be handled. This delaunizer also produces fewer mesh nodes than the previous algorithm. Reference [1] is an excellent book on Delaunay mesh generation.

The algorithm uses two independent structures to generate the final mesh:

- A set of surface faces (for example, the input boundary and some isosurfaces, or rectangular faces originating from user-defined refinement inside Sentaurus Mesh)
- A background 3D generic Delaunay triangulation

The algorithm works in the following way:

- Ridges and corners are classified.
- A set of protection spheres is generated around ridges and corners.
- A 2D surface delaunization algorithm is applied. This algorithm flips all nonridge edges that do not meet the Delaunay criterion.
- Each ridge that does not meet the Delaunay criterion is refined (see [2]).
- Each surface face that does not meet the Delaunay criterion is refined (see [2]).
- Each element that does not meet the quality criteria is refined.
- Surface faces that have not been recovered by refinement are recovered using a constrained Delaunay triangulation (CDT) algorithm (see [3][4]).
- Slivers are eliminated (see [5][6]).
- A material is assigned to each tetrahedron in the final triangulation.

Generating Ridges and Corners

During the first stage, the algorithm detects the faces that are coplanar. Every edge that bounds a coplanar set of faces is labeled a *ridge*. Every point that connects two non-collinear ridges is labeled a *corner*.

By default, two faces are coplanar if the angle between them is less than `coplanarityAngle` and the surface deformation that can result from flipping the common edge is less than `coplanarityDistance`.

Protecting Ridges and Corners

In general, conforming Delaunay algorithms do not perform well if the input contains sharp angles between adjacent faces on the surface (in general, of less than 60°). A generic algorithm would refine excessively around ridges and corners that define a very sharp angle. Occasionally, generic algorithms do not stop and the algorithms collapse.

The algorithm for ridge and corner refinement carefully refines around sharp ridges and corners, defining a set of spheres that protect these entities. Refinement points that are inside these spheres snap to the surface of the sphere. This produces constructions that resemble isosceles triangles, which are well suited to Delaunay-type algorithms because isosceles triangles contain their circumscribed centers inside them.

Conforming Delaunay Triangulation Algorithm

The conforming Delaunay triangulation (CDT) algorithm enables the delaunizer to produce meshes that are near-Delaunay after relaxing the Delaunay criterion.

After the faces have been refined to meet the (possibly relaxed) Delaunay criterion, some surface faces might be missing from the background 3D Delaunay mesh. The CDT algorithm inserts those faces into the background triangulation using a sequence of 3D face flips.

This algorithm is very complex, so you can expect long runtimes if the Delaunay criterion is relaxed too much at locations with many faces to be recovered (such as locations with a lot of refinement in Sentaurus Mesh).

Optimizing Elements

After the CDT algorithm is finished, the quality of the elements in the mesh might not be optimal. Therefore, the algorithm performs an extra refinement step, which eliminates all elements that do not meet the quality criteria specified by users. The quality criteria are:

- The maximum solid angle inside an element
- The maximum ratio between the circumscribed spheres of neighboring elements

Any element that does not meet the quality criteria is refined. The algorithm used to refine the elements is based on the Delaunay refinement technique, which inserts a node at the Voronoï center of the element and updates the neighboring triangulation. If the Voronoï center of the element lies too close to the surface, the surface is refined.

Eliminating Slivers

The last step in the delaunization involves eliminating sliver elements. To perform this, the algorithm uses a variation of the sliver exudation technique, which assigns weights to the nodes in the triangulation and uses them to compute a weighted Delaunay triangulation. The weights are increased selectively to eliminate slivers locally in the triangulation.

The sliver elimination step changes the regular Voronoï diagram, producing Voronoï cells that have negative sides.

The amount of damage is proportional to the weight applied to the mesh nodes. Therefore, the algorithm includes the parameter `sliverDistance`, which controls the amount of damage to the mesh. This parameter represents the maximum weight applied to a mesh node.

References

[1] S.-W. Cheng, T. K. Dey, and J. R. Shewchuk, *Delaunay Mesh Generation*, Boca Raton, Florida: CRC Press, 2013.

[2] J. R. Shewchuk, “Mesh Generation for Domains with Small Angles,” in *16th Annual Symposium on Computational Geometry*, Hong Kong, pp. 1–10, June 2000.

[3] J. R. Shewchuk, “Constrained Delaunay Tetrahedralizations and Provably Good Boundary Recovery,” in *Proceedings of the 11th International Meshing Roundtable*, Ithaca, NY, USA, pp. 193–204, September 2002.

[4] J. R. Shewchuk, “Updating and Constructing Constrained Delaunay and Constrained Regular Triangulations by Flips,” in *19th Annual Symposium on Computational Geometry*, San Diego, CA, USA, pp. 181–190, June 2003.

[5] S.-W. Cheng *et al*., “Sliver Exudation,” *Journal of the ACM*, vol. 47, no. 5, pp. 883–904, 2000.

[6] H. Edelsbrunner and D. Guoy, “An Experimental Study of Sliver Exudation,” in *Proceedings of the 10th International Meshing Roundtable*, Newport Beach, CA, USA, pp. 307–316, 2001.

A

Formulas for Analytic Profiles

This appendix presents general concepts and the models available along the primary direction and the lateral direction.

Sentaurus Mesh implements a complete set of analytic models to describe a wide range of different situations. The reason for implementing analytic profiles is to have a flexible tool to substitute process simulation results efficiently and within a reasonable time.

Although the formulas are designed according to the models associated with impurity concentrations, analytic profiles can be used for any type of variable defined in the output files.

General Concepts

Impurity concentrations can be represented by a set of 1D, 2D, and 3D analytic models. To describe each analytic model, two main directions must be defined:

- The *primary direction* that is perpendicular to the reference region
- The *lateral direction* that is parallel to the reference region

Along each direction, one function is defined, that is, the *primary function* and *lateral function*. The correct combination of both functions allows you to have an analytic description of a species concentration.

Local Coordinate Systems, Valid Domains, and Reference Regions

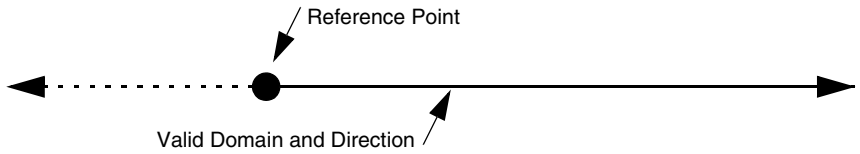
The valid domain for the analytic models depends on the reference region, which is defined using a dimension-dependent geometric element, and it is placed along the lateral direction.

By combining the reference region and primary direction, you can define a local coordinate system for each analytic function.

One-Dimensional Profiles

One-dimensional profiles require only the definition of the primary function, which is applied along the x-axis. The primary direction and valid domain are defined using a vector. The reference region for a profile is defined by using a geometric element, that is, a point. Figure 50 shows the scheme used for the 1D case.

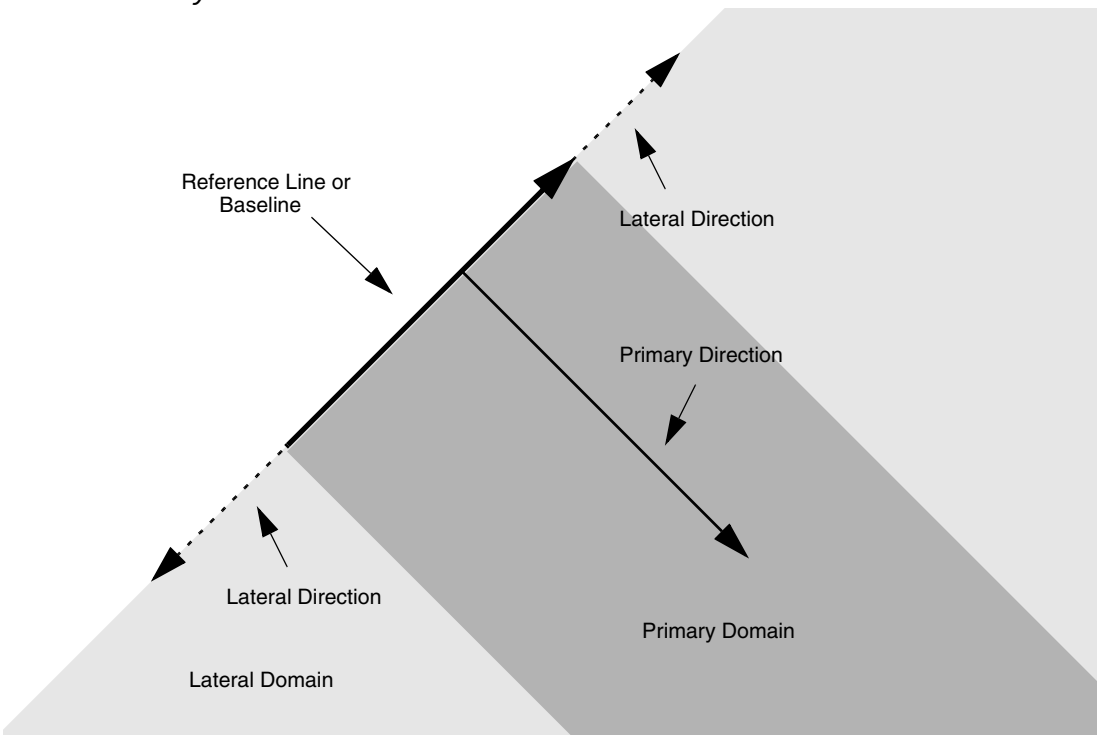
Figure 50 Primary direction in one dimension



Two-Dimensional Profiles

For 2D profiles, the reference region is defined using a *baseline*. The primary direction is the normal vector to the baseline and the lateral direction is parallel to the baseline. Figure 51 shows the general scheme of the local coordinate system and the valid domain. The valid domain for both the primary and lateral functions is defined by sweeping the primary direction vector along the lateral direction.

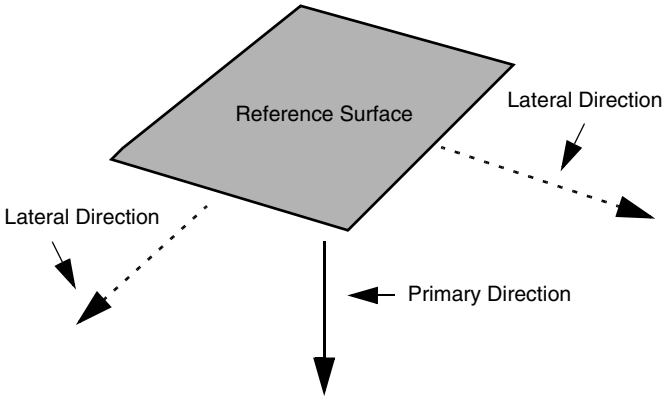
Figure 51 Primary and lateral directions in two dimensions



Three-Dimensional Profiles

For 3D profiles, the reference region is defined using a *surface*. The primary direction is the normal vector to the surface and the lateral direction is the plane perpendicular to the primary direction. Figure 52 shows the general scheme of the local coordinate system and the valid domain. The valid domain for both primary and lateral functions is defined by sweeping the primary direction vector along the surface.

Figure 52 Primary and lateral directions in three dimensions



General Implantation Models

In general, impurity concentrations can be expressed as:

$$\text{doping}(\vec{x}_p, \vec{x}_l) = g\langle \vec{x}_p \rangle \cdot f\langle \vec{x}_l \rangle \quad (3)$$

where:

- $g\langle \vec{x}_p \rangle$ represents the primary function in the local coordinate system.
- $f\langle \vec{x}_l \rangle$ represents the lateral function in the local coordinate system.

The most important functions used as models are Gaussian functions and error functions. For the remainder of this appendix, functions along the primary direction are referred to as $g\langle y \rangle$ and functions along the lateral direction, as $f\langle x \rangle$. The indices *y* and *x* are important to distinguish parameters among the different directions.

Each model is defined by a minimum set of parameters. This section presents a basic formulation of each model by using the minimum set of parameters. Subsequent sections show how to obtain this minimum set from different input or initial conditions.

Gaussian Function

The minimum set of parameters to define a Gaussian function is:

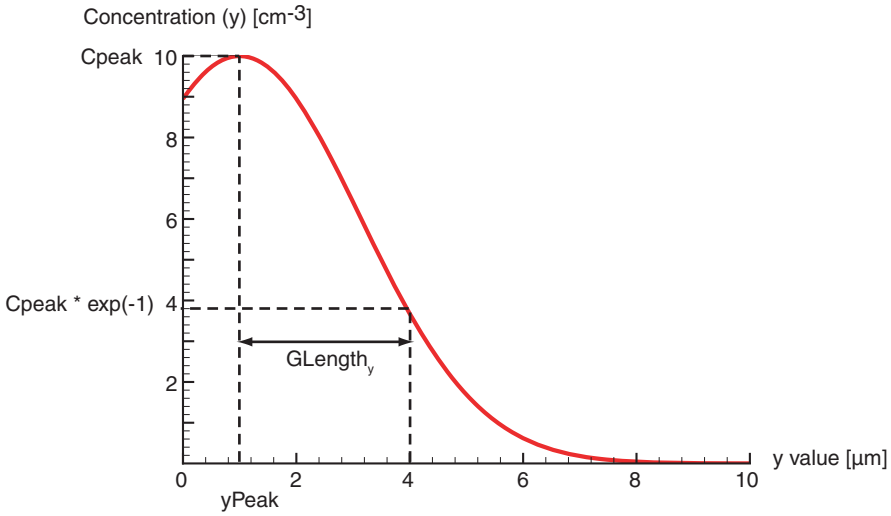
- Peak concentration (C_{peak}) [cm^{-3}]
- Peak position (y_{peak}) [µm]
- Length (GLength_y) [µm] or standard deviation (stdDev_y) [µm]

Using these parameters, the Gaussian function is defined by:

$$g(y) = C_{\text{peak}} \cdot \exp\left(-\frac{1}{2} \cdot \left[\frac{y - y_{\text{peak}}}{\text{stdDev}_y}\right]^2\right) = C_{\text{peak}} \cdot \exp\left(-\left[\frac{y - y_{\text{peak}}}{\text{GLength}_y}\right]^2\right) \quad (4)$$

Figure 53 shows the model schematically.

Figure 53 General shape of Gaussian functions



Error Function

The minimum set of parameters to define an error function as a doping profile is:

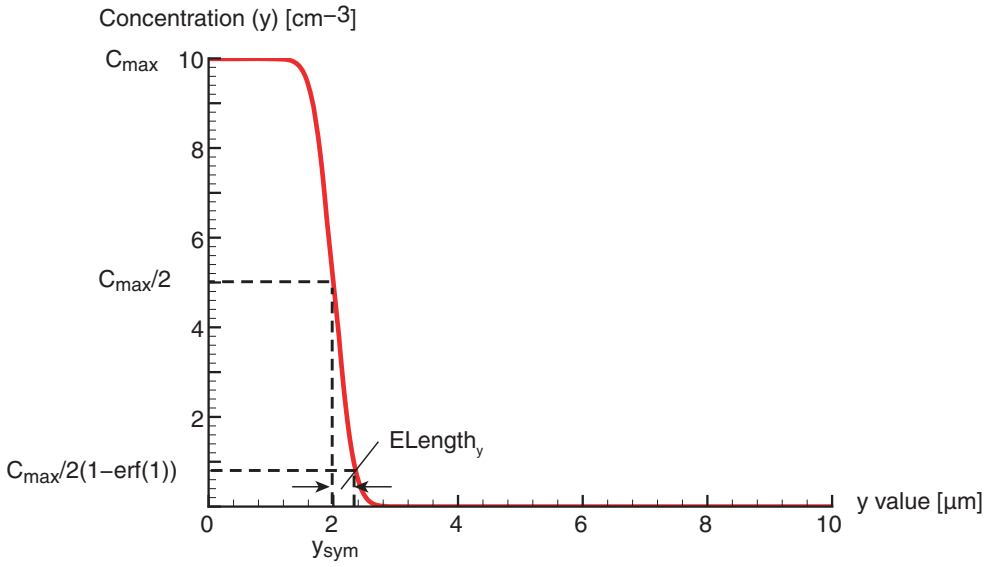
- Maximum concentration (C_{max}) [cm^{-3}]
- Symmetry position (y_{sym}) [µm]
- Length (ELength_y) [µm]

Using these parameters, the error function is defined by:

$$g(y) = \frac{C_{\max}}{2} \cdot \left(1 + erf\left[\frac{y_{\text{sym}} - y}{\text{ELength}_y}\right]\right) = \frac{C_{\max}}{2} \cdot \left(1 - erf\left[\frac{y - y_{\text{sym}}}{\text{ELength}_y}\right]\right) \quad (5)$$

The function is symmetric with respect to the inflection point.

Figure 54 General shape of error functions



Other Relevant Parameters

To have flexible models, some special parameters must be considered. These are not included in the standard formulation. However, by applying some definitions, the basic set can be obtained from them.

Dose

From a process simulation perspective, implantation functions are determined giving the dose concentration of the profiles. The peak concentration value can be obtained from the *Dose* (see Available Models Along the Primary Direction on page 155). Dose is given in atoms per cm^{-2}.

The general definition of Dose is:

$$\text{Dose} = \int_0^{\infty} g(y)\,dy \quad (6)$$

For Gaussian functions, the Dose is represented as:

$$\text{Dose} = \int_0^{\infty} C_{\text{peak}} \cdot \exp\left(-\frac{1}{2} \cdot \left[\frac{y - y_{\text{peak}}}{\text{stdDev}_y}\right]^2\right) dy \tag{7}$$

$$\text{Dose} = \frac{C_{\text{peak}} \cdot \sqrt{\pi} \cdot \text{stdDev}_y}{\sqrt{2}} \cdot \left(1 + erf\left[\frac{y_{\text{peak}}}{\sqrt{2} \cdot \text{stdDev}_y}\right]\right) \tag{8}$$

For error functions, the Dose is defined as:

$$\text{Dose} = \int_0^{\infty} \frac{C_{\max}}{2} \cdot \left(1 + erf\left[\frac{y_{\text{sym}} - y}{\text{ELength}_y}\right]\right) dy \tag{9}$$

$$\text{Dose} = \frac{C_{\max} \cdot \text{ELength}_y}{2} \cdot \left(\frac{y_{\text{sym}}}{\text{ELength}_y} \cdot \left[1 + erf\left(\frac{y_{\text{sym}}}{\text{ELength}_y}\right)\right] + \frac{1}{\sqrt{\pi}} \cdot \exp\left[-\left(\frac{y_{\text{sym}}}{\text{ELength}_y}\right)^2\right]\right) \tag{10}$$

Values at the Junction

The *Junction Concentration* and *Depth* parameters define either Gaussian or error functions. For a description of these parameters and how they can replace the standard deviation in the basic formulation, see Available Models Along the Primary Direction.

Length

For Gaussian functions, GLength represents the distance between the peak position and a place where the concentration decays by a factor of exp(–1) (36%) with respect to the peak concentration (see Figure 53 on page 153). The relationship between the length and the standard deviation for Gaussian functions is:

$$\text{GLength}_y = \sqrt{2} \cdot \text{stdDev}_y \tag{11}$$

Available Models Along the Primary Direction

The following models in Sentaurus Mesh are applied along the primary direction:

- Gaussian Functions
- Error Functions
- Constant Functions
- 1D External Profiles

Gaussian Functions

The basic set for Gaussian functions is formed by C_{peak}, y_{peak}, and stdDev_y. According to user input, the basic set of parameters can be specified in different ways depending on the parameters used to calculate C_{peak} and stdDev_y:

- Peak Concentration and Standard Deviation

  The basic set is complete (see Equation 4) and no basic parameters are computed.

- Peak Concentration and Length

  Standard Deviation is computed from GLength using:

$$\text{stdDev}_y = \frac{\text{GLength}_y}{\sqrt{2}} \tag{12}$$

- Dose and Standard Deviation

  Given Dose and Standard Deviation, the Peak Concentration value is calculated using:

$$C_{\text{peak}} = \frac{\text{Dose} \cdot \text{factor} \cdot \sqrt{2}}{\sqrt{\pi} \cdot \text{stdDev}_y \cdot \left(1 + erf\left[\frac{y_{\text{peak}}}{\sqrt{2} \cdot \text{stdDev}_y}\right]\right)} \tag{13}$$

  where $\text{factor} = 10^4$ because Dose is in cm^{-2}.

- Dose and Length

  Given Dose and GLength, the Standard Deviation is computed from Equation 12, and the Peak Concentration is computed from Equation 13.

- Peak Concentration and values at the junction

  Standard Deviation is computed from the values at the junction using:

$$\text{stdDev}_y = \frac{y_{\text{depth}} - y_{\text{peak}}}{\sqrt{-2 \cdot \ln(C_{\text{atDepth}} / C_{\text{peak}})}} \tag{14}$$

  Note:
  C_{peak} must be greater than C_{atDepth}.

- Dose and values at the junction

  First, Standard Deviation is computed from:

$$\frac{C_{\text{atDepth}} \cdot \sqrt{\pi} \cdot \text{stdDev}_y \cdot \left(1 + erf\left[\frac{y_{\text{peak}}}{\sqrt{2} \cdot \text{stdDev}_y}\right]\right)}{\sqrt{2} \cdot \text{Dose} \cdot \text{factor}} = \exp\left(-\frac{1}{2} \cdot \left[\frac{y_{\text{depth}} - y_{\text{peak}}}{\text{stdDev}_y}\right]^2\right) \tag{15}$$

Second, using stdDev_y, Peak Concentration is computed as in C.

Note:
Equation 15 is an implicit equation and Dose is in cm^{-2}.

Error Functions

For error functions, the basic set of parameters includes $C_{\max}$, y_{sym}, and $\mathrm{ELength}_y$ and can be computed in different ways:

- Maximum Concentration and Length

  The basic set is complete and no parameters are computed (see Equation 5).

- Dose and Length

  Maximum Concentration is computed from Dose using:

$$C_{\max} = \frac{2 \cdot \mathrm{Dose}}{\mathrm{ELength}_y} \cdot \mathrm{factor} \cdot \left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \cdot \left[1 + erf\left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \right) \right] + \frac{1}{\sqrt{\pi}} \cdot \exp\left[-\left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \right)^2 \right] \right)^{-1} \quad (16)$$

  where $\mathrm{factor} = 10^4$ because Dose is in cm^{-2}.

- Maximum Concentration and values at the junction

  ELength can be computed from:

$$erf\left(\frac{y_{\mathrm{sym}} - y_{\mathrm{depth}}}{\mathrm{ELength}_y} \right) = \frac{2 \cdot C_{\mathrm{atDepth}}}{C_{\max}} - 1 \quad (17)$$

  Note:
  Equation 17 is an implicit equation.

- Dose and values at the junction

  Maximum Concentration and ELength are computed using the following implicit equations, which follow from Equation 16 and:

$$\mathrm{Dose} \cdot \mathrm{factor} \cdot \left(1 + erf\left[\frac{y_{\mathrm{sym}} - y_{\mathrm{depth}}}{\mathrm{ELength}_y} \right] \right) = C_{\mathrm{atDepth}} \cdot \mathrm{ELength}_y \cdot \left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \cdot \left[1 + erf\left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \right) \right] + \frac{1}{\sqrt{\pi}} \cdot \exp\left(-\left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \right)^2 \right) \right) \quad (18)$$

$$C_{\max} = \frac{2 \cdot \mathrm{Dose}}{\mathrm{ELength}_y} \cdot \mathrm{factor} \cdot \left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \cdot \left[1 + erf\left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \right) \right] + \frac{1}{\sqrt{\pi}} \cdot \exp\left[-\left(\frac{y_{\mathrm{sym}}}{\mathrm{ELength}_y} \right)^2 \right] \right)^{-1} \quad (19)$$

Constant Functions

Constant functions are useful to define substrate doping mathematically:

$$g(y) = \text{Constant} \tag{20}$$

1D External Profiles

Real 1D process simulation results can be read along the primary direction. To complete the 2D profile and 3D profile, you add an analytic lateral function.

The values that do not appear in the file are interpolated using an interpolation function. Every species has a corresponding interpolation function predefined in the `datexcodes.txt` file. These functions can be linear, arsinh, or logarithmic.

If h is an interpolation function, then the value at point y is computed from an external 1D profile as follows:

$$g(y) = \begin{cases} \text{data}_i & y = y_i \\ h^{-1}\left(\frac{y-y_i}{y_{i+1}-y_i} \cdot h(\text{data}_{i+1}) + \frac{y-y_{i+1}}{y_{i+1}-y_i} \cdot h(\text{data}_i)\right) & y_i < y < y_{i+1} \end{cases} \tag{21}$$

Lateral or Decay Functions

Lateral or decay functions are evaluated on the valid lateral domain (see Figure 51 on page 151 and Figure 52 on page 152). They are defined as the decay along the lateral direction and depend on the distance from the valid primary domain of the point to evaluate.

For 2D simulations, this distance is calculated using the baseline as reference. For 3D simulations, the distance is computed using the surface as reference. The following models are available:

- Lateral Gaussian Function
- Lateral Error Function
- No Lateral Function

Note:

Lateral or decay functions are not valid for 1D simulations.

Lateral Gaussian Function

The equation applied is:

$$f(x) = \exp\left(-\frac{1}{2} \cdot \left[\frac{x_{\text{closest}P} - x}{\text{stdDev}_x}\right]^2\right) \tag{22}$$

According to Equation 22, the required value from the user is the standard deviation, stdDev_x, along the lateral direction. You can define it in one of the following ways:

- Provide the value explicitly
- Provide a factor with respect to the standard deviation along the primary direction:

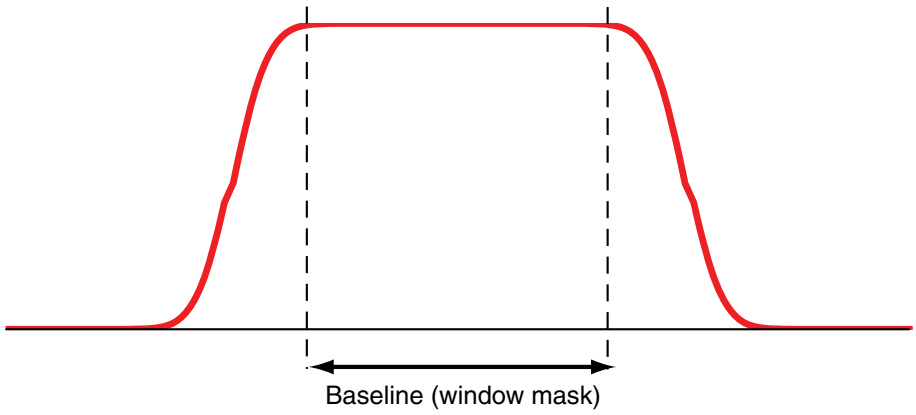
$$\text{stdDev}_x = \text{Factor}_x \cdot \text{stdDev}_y \tag{23}$$

- Give the length of the Gaussian function:

$$\text{GLength}_x = \text{stdDev}_x \cdot \sqrt{2} \tag{24}$$

Using this function, the decay begins outside the primary domain, that is, the overlap between the primary, lateral, and decay domains is zero. Figure 55 shows this effect.

Figure 55 Using Gaussian function as lateral function in two dimensions



Lateral Error Function

By default, when specifying an error function as a lateral function in an analytic profile, the following equation is applied:

$$f(x) = \frac{1}{2} \cdot \left(1 + erf\left[\frac{x_{\text{closest}P} - x}{\text{ELength}_x}\right]\right) \tag{25}$$

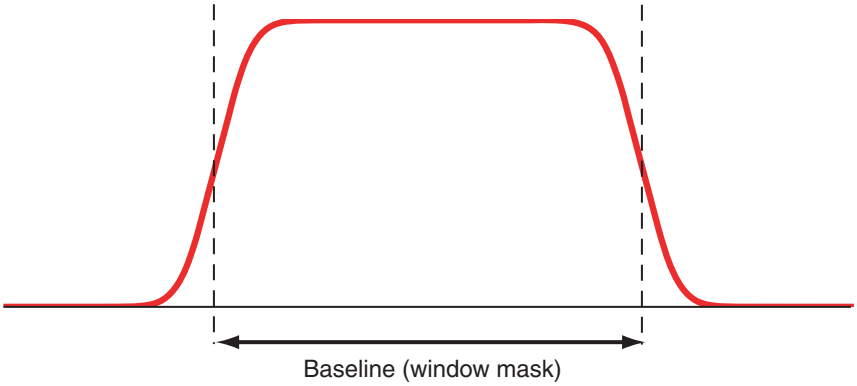
According to Equation 25, the required value from the user is the length for the error function, $\mathrm{ELength}_x$, along the lateral direction. You can define it in one of the following ways:

- Provide the value explicitly
- Provide a factor with respect to the length along the primary direction:

$$\mathrm{ELength}_x = \mathrm{Factor}_x \cdot \mathrm{ELength}_y \tag{26}$$

For this model, the overlap of the primary, lateral, and decay domains is not zero. The lateral decay starts inside the primary domain as shown in Figure 56.

Figure 56 Using error function as lateral function in two dimensions



In some situations, the previous formulation can lead to a change in the total dose defined by the analytic profile. In those cases, you can use the `lateralDiffusion` parameter in the `Interpolate` section of the Sentaurus Mesh command file to change the formula used to calculate the error function (see Interpolate Section on page 41):

$$f(x) = \frac{1}{2} \cdot \left(erfc\left[\frac{x - x_{\max}}{\mathrm{ELength}_x} \right] - erfc\left[\frac{x - x_{\min}}{\mathrm{ELength}_x} \right] \right) \tag{27}$$

For higher dimensions:

$$f(x, y, z) = f(x) \cdot f(y) \cdot f(z) \tag{28}$$

These formulas are used only if the reference element is rectangular. In this case, $x_{\min}$ and $x_{\max}$ are the minimum and the maximum coordinates of the reference element, respectively.

No Lateral Function

This property is valid when Factor is equal to zero. In this case, the value of the lateral function is given by:

$$f(x) = \begin{cases} 1 & x \in \text{PrimaryDomain} \\ 0 & x \notin \text{PrimaryDomain} \end{cases} \quad (29)$$

The lateral domain is null.

B

Doping Function for Discrete Dopants

This appendix describes the doping function used to transform discrete dopants into a continuous doping profile.

Sentaurus Mesh can create continuous doping profiles from discrete dopant distributions obtained from Sentaurus Process Kinetic Monte Carlo (Sentaurus Process KMC). This is accomplished by associating a doping function with each discrete dopant. The union of all such doping functions defines the final doping profile for the structure.

Doping Function

It has been suggested [1] that the charge density associated with a discrete dopant be can separated into short-range and long-range portions, and that the long-range portion is appropriate for inclusion in drift-diffusion device simulators. Sentaurus Mesh uses the long-range portion of the number density associated with a discrete dopant suggested in [1]:

$$n(r) = N_f \cdot \frac{k_c^3}{2\pi^2} \frac{\sin(k_c r) - (k_c r)\cos(k_c r)}{(k_c r)^3} \tag{30}$$

In this expression:

- r is the distance from the discrete dopant.
- k_c is the inverse of the screening length.
- N_f is a normalization factor such that the integral of $n(r)$ over the entire simulation space becomes unity.

Note that this expression is oscillatory and becomes negative for certain values of $k_c r$ (see Figure 57 on page 163).

In Sentaurus Mesh, however, this expression is cut off at the first zero of $n(r)$, that is:

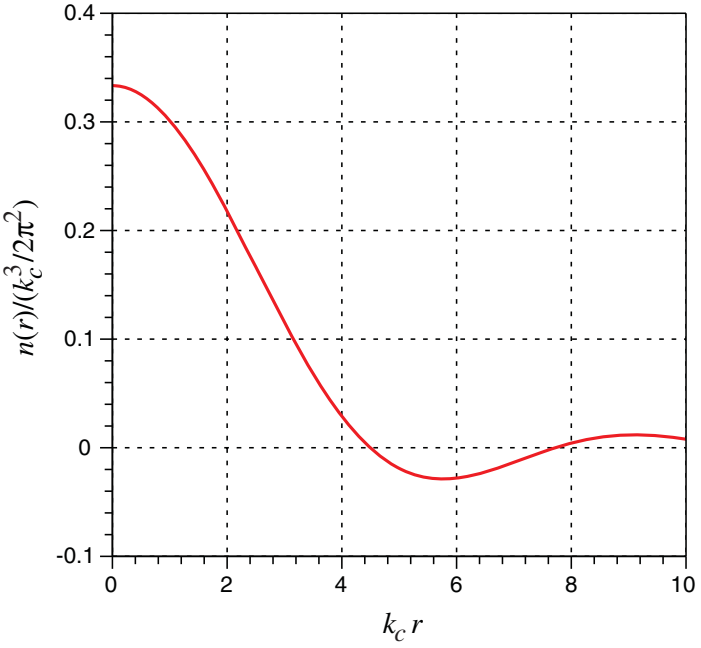
$$\begin{aligned} n(r) &= N_f \cdot \frac{k_c^3}{2\pi^2} \frac{\sin(k_c r) - (k_c r)\cos(k_c r)}{(k_c r)^3} \quad , \quad k_c r < 4.4934 \\ &= 0 \quad , \quad k_c r \geq 4.4934 \end{aligned} \tag{31}$$

In this case, the normalization factor is taken to be N_f= 0.59688.

Note:

This normalization factor assumes that the function given in Equation 31 does not extend outside the simulation space. In general, this is not true for discrete dopants located near boundaries.

Figure 57 Long-range number density associated with a discrete dopant



Cut-off Parameter

The inverse of the cut-off parameter, $1/k_c$, is the screening length. Different charge screening models suggest different expressions for k_c as a function of impurity concentration. One model suggests that k_c is the inverse of the Debye length:

$$k_c = \sqrt{\frac{Ne^2}{\varepsilon_{Si} k_B T}} \tag{32}$$

where:

- N is the impurity concentration.
- e is the electronic charge.
- $\varepsilon_{\mathrm{Si}}$ is the permittivity of silicon.
- k_{B} is the Boltzmann constant.
- T is the temperature.

However, the paper [1] prefers a charge screening model that gives k_c simply as:

$$k_c \approx 2N^{1/3} \tag{33}$$

In practice, k_c can be used as a fitting parameter, for example, by comparing the threshold voltage for a large MOSFET when the present doping model is used with the threshold voltage obtained with a standard continuum doping model.

In Sentaurus Mesh, k_c is a user-adjustable parameter called `ScreeningFactor`. It is part of the `Particle` definition and it can be specified separately for each `Species`.

References

[1] N. Sano *et al*., “On discrete random dopant modeling in drift-diffusion simulations: physical meaning of ‘atomistic’ dopants,” *Microelectronics Reliability*, vol. 42, no. 2, pp.189–199, 2002.