User Guide v-1

Structure Generator and Mesher User Guide

SemiVi LLC

Zollikon, Switzerland.

November 3, 2025

Contents

Chapter 1

Introduction

The structure generator and mesher tool is used to create and mesh semiconductor device structures for various optoelectronic simulations. The user can create a config file which contains information on the device structure, doping, and materials. This config file is parsed by the software to create and mesh the device structure and store it in hdf5 format. A xdmf script file is also generated for viewing the device structure and mesh in paraview.

1.1 Features

The structure generator supports creating device with various common shapes (see Chap. 3). These shapes are then meshed using different types of meshers suitable for various device simulators provided. Following mesh are supported.

- FEM-mesh: A triangular (2D devices) or tetrahedral (3D devices) finite-elements based mesh. It is used for electrical drift-diffusion transport simulations, and for dopant diffusion process simulations.

- Tensor-mesh: A rectangular (2D devices) or cubic (3D devices) grid based mesh. It is used for optical Finite Difference Time Domain (FDTD) simulations, Beam Propagation Method (BPM)

simulations. It is also used for electrical Quantum Transport (QT) simulations with Non-equilibrium Green's Function (NEGF) formalism.

- *Triangle* mesh: An external mesh engine which meshes 2D device with triangular finite-elements.
- *NetGen* mesh: An external mesh engine which meshes 3D device with tetrahedral finite-elements.

Since the structure generation and meshing have been separated, the same structure config file can be conveniently used to create any of the meshes by specifying arguments.

The structure generator also supports importing masks polygons from *GDS mask files* (Chap. 7) and use them to define regions or analytic doping profiles.

1.2 Installation

SemiVi currently supports software installation on various Linux distributions. The software installer is available in Debian package (*.deb file) and in RPM format (*.rpm file).

Download the installer on the local machine. The installer file named `mesher_amd64.deb` will appear in the `Downloads` directory. Go to the directory using `cd` command. Use the following command to install the `Mesher` from the installer.

```
>> sudo apt install ./mesher_amd64.deb
```

Alternately, one may use `dpkg` to install the software and use `apt` to install missing dependencies as follows.

```
>> sudo dpkg -i ./mesher_amd64.deb
>> sudo apt install -f
```

You need to have `root` access to install the software on your machine.

1.3 Licensing

Two types of licenses can be purchased for SemiVi Mesher.

Node-locked licenses enable unlimited number of *simultaneous* executions of the Mesher on the client machine. The node-locked license limits the usage of the Mesher only to the machine on which the license is activated.

With *server licenses*, the Mesher can be run on any of the machines in the client organization on which the server license is activated. However, only the specified number of *simultaneous* executions are possibie at a time.

1.3.1 Purchasing the licenses

The clients can place order for any of the above licenses on SemiVi website (https://www.semivi.ch/sales) or by contacting our sales-person.

We will process the request and send the license files by email. The license files need to be activated on the desired machines using the license key which is emailed separately using the following command.

1.3.2 Installation of SemiVi-activator

The license file must be activated on the desired computer before use. For that purpose, download the installer `semivila_amd64.deb` file on the local machine and install it as follows.

```
>> sudo apt install ./semivila_amd64.deb
```

1.3.3 License activation

To activate the license file, please run the following command.

```
>> semivila -a File.lic <Server|NodeLocked>License.lic\\
```

Replace `File.lic` with the your license file, and use appropriate name for the activated file. You will be prompted to input the 16 digit license key. A successful activate of the license file will generate the activated license file. Copy the activated license file to the

`/opt/semivi/licenses/` folder and rename it to `ServerLicense.lic` or `NodeLockedLicense.lic` for server and node-locked licenses respectively. If you have more than one license files, please delete the older expired license files. If you wish to keep more than one active license files, you can also name the license files as `<i>NodeLockedLicense.lic` where `<i>` could be from 0 to 49. For ex. `49NodeLockedLicense.lic` or `49ServerLicense.lic`. The program will read the license files and lock the first available license. All the target users must have read rights on the license file.

User-guides of all the software provided by SemiVi are stored at the location `/opt/semivi/userguides/`.

Tutorials of all the software provided by SemiVi are stored at the location `/opt/semivi/tutorials/mesher`.

Chapter 2

Device Generation File Structure

The structure generator and mesher is provided with every software package of SemiVi. The structure generator and mesher work together to create a device and passes it to the simulator. A standalone 'Mesher' program can also be invoked to create and save the device in hdf5 format and view it in paraview using xdmf file. Here, the standalone program is used to create the device using the following command - `>> Mesher str diode_str.cfg`

In the above command, the word after `Mesher` is the name of the program to be executed (in this case – `str`). The program name is followed by the device structure generation file name (in this case – `diode_str.cfg`).

2.1 Config File structure

A sample structure file which creates a 2D pn-diode is provided below.

```
Device:
{
```

```
Name = "Diode";
MeshType = "FEM";
Simulation = "DD";
}

RefWin*RefSi:
{
Position: ([-1.0, -0.2, 0.], [1., 0.2, 0.]);
Shape = "Rectangle";
}

RefWin*RefCathode:
{
Position: ([0.999, -0.25, 0.], [1.001, 0.25, 0.]);
Shape = "Rectangle";
}

RefWin*RefAnode:
{
Position: ([-0.999, -0.25, 0.], [-1.001, 0.25, 0.]);
Shape = "Rectangle";
}

RefWin*RefPDoping:
{
Position: ([-1.001, -0.25, 0.], [0., 0.25, 0.]);
Shape = "Rectangle";
}

RefWin*RefNDoping:
{
Position: ([0.0, -0.25, 0.], [1.001, 0.25, 0.]);
Shape = "Rectangle";
}

Region*RegSi:
{
RefWin = ["RefSi"];
```

```
Material = "Silicon";
//MaxEdgeLength = 0.05;
}

MeshDef*MDefSiTop:
{
RefWin = "RefAll";
Xarg: [ 0., 0.0001, 0.05, 1.3 ]
Yarg: [0., 0.025, 0.025, 1. ]
Zarg: [0., 0.025, 0.025, 1. ]
}

DopingDef*Ndoping:
{
RefWin = "RefNDoping"
Type = "Constant"
Concentration = 5E16
Dopant = "Phosphorus"
}

DopingDef*Pdoping:
{
RefWin = "RefPDoping";
Type =  "Constant"; // "Constant", "Linear", "Exponential", "Analytic'
Concentration = 5E16;
Dopant = "Boron";
}

Contact*Cathode:
{
RefWin = ["RefCathode"]
}

Contact*Anode:
{
RefWin = ["RefAnode"]
}
```

The above structure generation file is composed of various sections which define different objects of the device. In each of the object names, string before '*' gives the object type, whereas the string after '*' specifies the object name. Various keywords in each of the object and their functionality is shortly described below. Except for the `File` section, all the objects can be multiply define in a config file. However, each object must have a *different* name. If two objects of the same name are defined, the parser will give an error.

2.1.1 Device section

There can be only one `Device` section in the file. `Name` sets the device name. The generated mesh is stored in '<Name>_str.h5' file. A script file '<Name>_str.xdmf' is generated for viewing the device in paraview. `MeshType` sets that internal 'FEM' meshing engine to be used. `Simulation` provides the type of simulation to be performed using the device.

2.1.2 RefWin objects

Each `RefWin` object creates a specific shape set by `Shape` using the points defined as a `(...)` list of three floats in `[...]` bracket. Number of points and their meaning varies with the shape. An axis-aligned `Rectangle` shape is defined by two points. They form two diagonally opposite points of the rectangle object. A 'RefWin' has no meaning on its own, unless it is referenced by any other object such as 'Region' or 'MeshDef'. One RefWin object can be referenced by many Region, Doping, Contact, or other objects.

2.1.3 Region objects

Each `Region` object defines a region with a specific material. It consists of a comma separated list of 'RefWin' names which constitute the region is provided with `RefWin` keyword. `Material` sets material of the region. Volume encompassed by the RefWins is a part of the region. Note, that the order in which the region objects are defined is important. If there is an overlap between two regions, the region defined latest in the config file gets precedence.

2.1.4 Doping objects

Each `Doping` object defines a new doping profile in the region. The doping object is linked to a RefWin object. The doping profile is defined by `Type`. Here, all the doping definitions have a constant profile. `Concentration` sets concentration of 'constant' profile. This means, all the vertices inside the RefWin are set to constant profile. More than one doping objects can overlap. In that case, doping concentrations are added in the overlap region. Dopant name is set by `Dopant`.

2.1.5 Contact objects

Each `Contact` object defines a contact which has the same name as name of the object. It contains a list of comma separated list of 'RefWin' names. All the vertices inside each of the RefWins are set to this contact object. In case of an overlap, the contact defined latest in the config file gets precedence.

2.1.6 MeshDef objects

Each `MeshDef` object sets parameters of the mesh generator program specific to the volume confined in the RefWin named `RefWin`. It also specifies the parameters which are used in defining grid along x, y, and z axis using the keyword `X`, `Y`, and `Z`. The grid are defined as follows.

Grid parameters

Each of the `X`, `Y`, and `Z` grid definitions contains minimum grid-point spacing set by `Xarg[1]`, maximum grid-point spacing set by `Xarg[2]`. The grid points along that axis have spacing between the these values. If `Xarg[3]` factor is more than 1.0, then minimum grid spacing is multiplied with `Xarg[3]` factor for every subsequent grid spacing starting from grid point at `Xarg[0]`.

2.2 Generating a device

In this section, the above config file is used to generate the pn-diode structure shown in Fig. 2.1. The structure is created using the command

```
>> Mesher str diode_str.cfg.
```

The above command will generate '<Name>_str.h5' file containing the mesh data, together with a script file '<Name>_str.xdmf' for visualization. If Paraview is installed on the machine, the xdmf file can be opened using the following command

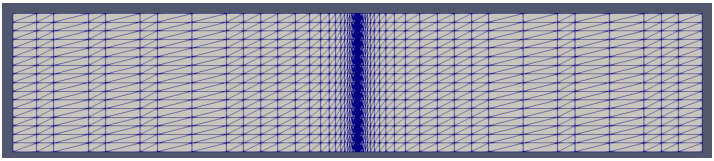
```
>> paraview Diode_str.cfg.
```

Once `paraview` is opened, see the `Pipeline Browser` window. Highlight `Diode_str.xdmf` file and click the button `Apply` below in `Properties` tab. You will see the device with spatial doping concentration. Now, find the drop-down menu written `Surface` and change it to `Surface With Edges`. Mesh will be displayed as shown in Fig. 2.1(a). Changing `DopingConcentration` to `Solid Color` shows the mesh as in Fig. 2.1(a). Now, select `VertexContactMap` instead of `DopingConcentration`. Also, open `Color Map Editor` and select the option `Interpret Values As Categories`. Vertices will be colored as per the contact they belong to (see Fig. 2.1(c)). Vertices which belong to `Anode` are colored green (for id 1), while those which belong to `Cathode` are colored red (for id 0).

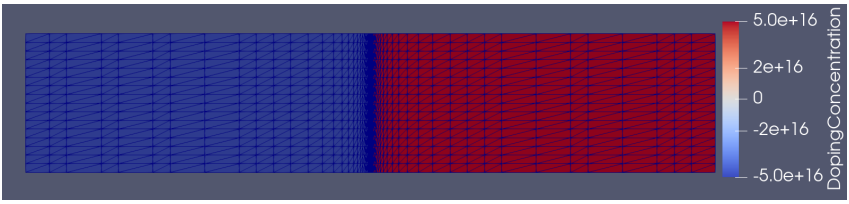
2.3 Modifying the device

Copy the above config file to any text editor and modify the mesh definition as follows.

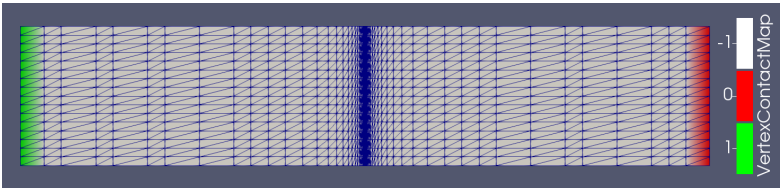
```
MeshDef*MDefSiTop:
{
  RefWin = "RefAll";
  Xarg: [ 0., 0.0001, 0.05, 1.3 ]
  Yarg: [0., 0.025, 0.025, 1. ]
  Zarg: [0., 0.025, 0.025, 1. ]
}
```



(a) Meshing

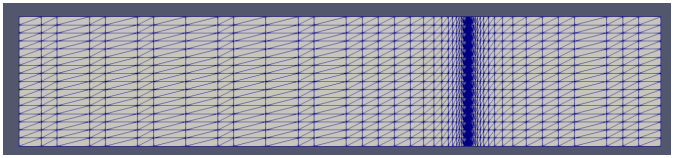


(b) Doping

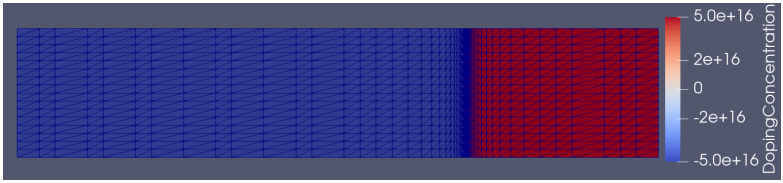


(c) Contacts

Figure 2.1: (a) Meshing generated by running the diode config file. (b) Doping concentration set by the config file. (c) Contact information set by the config file. Vertices are mapped to the contact they belong to. '-1' refers to the Semiconductor regions, '-2' refers to the insulator regions, and '-3' refers to the metal regions outside of any contact.



(a) Meshing



(b) Doping

Figure 2.2: (a) Meshing generated by running the modified diode config file. (b) Doping concentration set by the modified config file.

Run the meshing command.
`>> Mesher str diode_str.cfg`.
Mesh will be generated using the modified settings as shown in Fig 2.2(a). Now, modify doping of the device by changing the doping RefWin as follows.

```
RefWin*RefPDoping: {
  Position: ([-1.001, -0.25, 0.], [0.4, 0.25, 0.]);
  Shape = "Rectangle";
}

RefWin*RefNDoping: {
  Position: ([0.4, -0.25, 0.], [1.001, 0.25, 0.]);
  Shape = "Rectangle";
}
```

Rerun the meshing command. View the doping concentration. Now p-n junction is shifted to the right by 0.4μm as shown in Fig. 2.2(b). In this way, the meshing or doping can be modified. As an additional exercise, change the doping setting to the following.

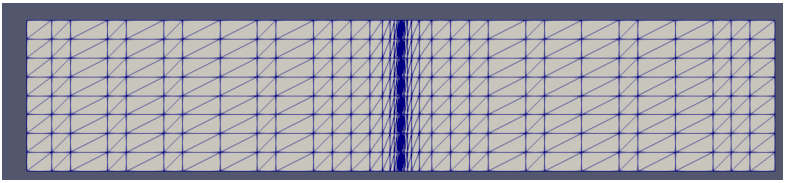


Figure 2.3: Meshing generated by running the modified diode config file by adding an addition refinement horizontally.

```
MeshDef*MDefSiTop:
{
  RefWin = "RefAll";
  Xarg: [ 0., 0.0001, 0.05, 1.3 ]
  Yarg: [0., 0.025, 0.025, 1. ]
  Zarg: [0., 0.025, 0.025, 1. ]
}
```

Rerun the mesher and view the generated mesh. By changing `Xarg[0]` from 1.0 to 1.7, a refined mesh is added horizontally as shown in Fig. 2.3.

2.4 Generating a 3D device

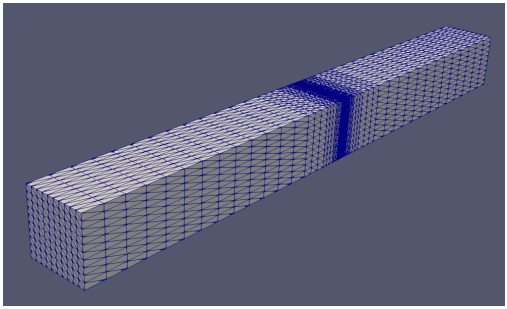
Copy the config file to another file and open it in a text editor. Edit *all* the `RefWin` objects in the file as follows.

```
RefWin*RefSi: {
  Position: ([-1.0, -0.2, -0.2], [1., 0.2, 0.2]);
  Shape = "Cuboid";
}
```

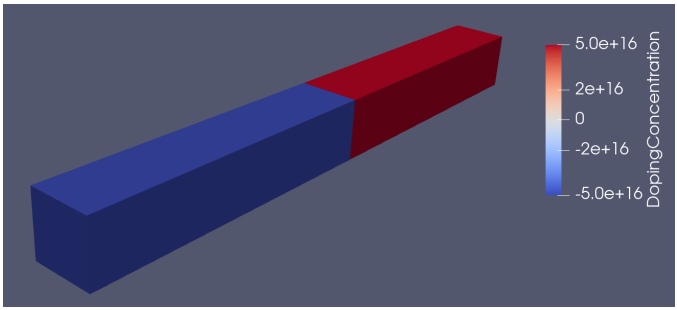
In the above snippet, z-coordinate of the first point in `Position` is changed from 0 to -0.2 and that of the second point is changed from 0 to 0.2. Also, its `Shape` was changed from `Rectangle` to `Cuboid`. Same changes are made to all the `RefWin` objects in the config file. This changes the diode from 2D structure to a 3D structure. The 3D structure can be meshed using the same command.

```
>> Mesher str diode_str.cfg.
```

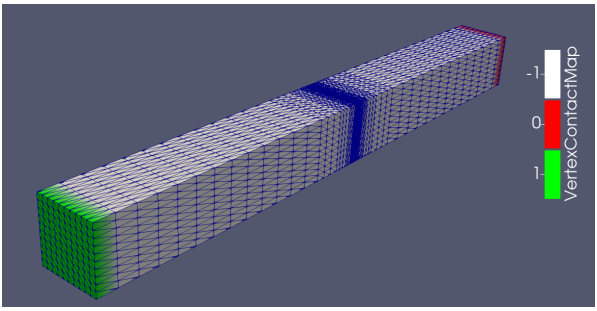
Resulting structure is stored in 'Diode_str.h5' file and the script is saved in 'Diode_str.xdmf' file. It can be opened in `paraview`. Same steps as given in Section 2.2 can be followed to view mesh as in Fig. 2.4(a), doping as in Fig. 2.4(b), and contact information as in Fig. 2.4(c).



(a) Mesh



(b) Doping



(c) Contacts

Figure 2.4: (a) Meshing generated by running the modified diode config file with 3D structure. (b) Doping concentration set by the config file. (c) Contact information set by the config file.

Chapter 3

Device Generation with Python Script

The structure generation and meshing can also be performed using a python script. This is performed by *device* library in python which interfaces python functions with the respective functions of the 'Mesher' program. If python interface for 'Mesher' is installed, any file containing python script to generate the device can be run like any other python file.

```
>> python diode_str.py
```

3.1 Python Script File structure

A sample python script file which creates a 2D pn-diode is provided below. The script produces the structure identical to that created by the config file in Sec. 2.1.

```
import mesher as m

# define device object
dio = m.device ("Diode", "FEM", "STR", 1)
```

```
if not dio.lockLicense () == 1:
print ("Could not fetch licenses.")
exit ()

xmin = -1.
xmax = 1.
xmid = 0.
ymin = -0.2
ymax = 0.2
# define RefWins
dio.setRefWin (Name="RefAll", Shape="Rectangle",
         Position=[m.position(xmin, ymin, 0.),
                   m.position(xmax, ymax, 0.)],
          NumericParams={},NumericListParams={})

dio.setRefWin (Name="RefSi1", Shape="Rectangle",
         Position=[m.position(xmin, ymin, 0.),
                   m.position(xmid, ymax, 0.)],
          NumericParams={},NumericListParams={})

win2 = m.refwin("RefSi2","Rectangle",
               [m.position(xmid, ymin, 0.),
                m.position(xmax, ymax, 0.)],{},{})

win2.saveSTL();

dio.addRefWinToDevice(win2)

dio.setRefWin (Name="RefCathode", Shape="Rectangle",
         Position=[m.position(xmin-1E-3, ymin-1E-3, 0.),
                   m.position(xmin+1E-3, ymax+1E-3, 0.)],
                NumericParams={},NumericListParams={})

dio.setRefWin (Name="RefAnode", Shape="Rectangle",
         Position=[m.position(xmax-1E-3, ymin-1E-3, 0),
                   m.position(xmax+1E-3, ymax+1E-3, 0.)],
                NumericParams={},NumericListParams={})
```

```
dio.setRefWin (Name="RefPDoping", Shape="Rectangle",
         Position=[m.position(xmin, ymin, 0.),
                   m.position(xmid, ymax, 0.)],
                NumericParams={},NumericListParams={})

dio.setRefWin (Name="RefNDoping", Shape="Rectangle",
         Position=[m.position(xmid, ymin, 0.),
                   m.position(xmax, ymax, 0.)],
                NumericParams={},NumericListParams={})

# define Regions
dio.setRegion (Name="RegSi1", RefWinName="RefSi",
               Material="Silicon", MeshingParams={})

dio.addRegion (Name="RegSi2", refwin=win2,
               Material="Silicon", MeshingParams={})

# define Meshing
dio.setMeshDef (Name="MDefAll", RefWinName="RefAll",
               Xarg=[0., 1E-4, 0.05, 1.3],
               Yarg=[0., 0.025, 0.025, 1.],
               Zarg=[0., 0.025, 0.025, 1.],
               FEMDopingGradientCutoff=1E30)

# define Doping
dio.setDopingDef (Name="Ndoping", RefWinName="RefNDoping",
               Type="Constant", Concentration=5E16,
               DopantSpecies ="Phosphorus")

dio.setDopingDef (Name="Pdoping", RefWinName="RefPDoping",
               Type="Constant", Concentration=5E16,
               DopantSpecies ="Boron")

# define Contacts
dio.setContact (Name="Anode", RefWinName="RefAnode")

dio.setContact (Name="Cathode", RefWinName="RefCathode")
```

```
# create Mesh
dio.createDeviceMesh ()

# save Mesh
dio.saveMeshData ()
```

First line of the python file `import device as dev` imports device library which has the device object called `Device`. The 'device' object contains all the procedures (python equivalent of functions) that are used to create a device structure and mesh it. Once a 'device object' is instantiated, these procedures can be invoked on the device instance as shown on subsequent lines of the file. Usage of each of the procedure is described below.

3.1.1 Initialization

`dio = dev.Device ("Diode", "FEM", "STR", 1)` line instantiates a device object 'dio'. This is similar to `File` section in config file and is described in Subsec. 2.1.1. It takes device name, mesh type, simulation type as input which correspond to `Name`, `MeshType`, and `Simulation` in `File` section. The last input is an integer 0 or 1 which tell the simulator if the object will be used for simulation ('1') or only for structure and mesh generation ('0'). If '0', then the generator does not calculate coupling element matrix which saves computation time.

3.1.2 Defining RefWin

`dio.setRefWin (...)` procedure is invoked on 'dio' object which adds a 'RefWin' to the object. Inputs are similar to the `RefWin` section in config file as described in Subsec. 2.1.2. Note, that `Position` argument takes coordinates in the form of a 'numpy' 2D array. The leading dimension has the size of number of points and the trailing dimension has the size of 3, corresponding to X, Y, and Z coordinates of the points, as shown in the above file. Note, that the above procedure adds `RefWin` to the device.

Also, the name of the RefWin is unique to it. Adding multiple RefWins with the same `Name` gives an error message and the RefWin is not added.

A RefWin can be referenced by many Region, Doping, Contact, or other objects using its unique name.

Alternately, a `RefWin` instance is created on the line `win2=m.refwin(...)`. It is named `win2`. This `RefWin` exists *outside* the device definition. This `RefWin` can be added to the device `dio` using the procedure, `dio.addRefWinToDevice(...)`. Once added to device, this `RefWin` can be referred by its name while adding Region, etc.

Various geometric operations can be performed on the `RefWin` object, e.g. rounding edges, boolean operations, sweeps, etc. The `RefWin` object is stored as an STL file (<RefWinName>.stl) using the procedure `win2.saveSTL()`. The STL file can be visualized using `paraview` to check geometric correctness.

3.1.3 Defining Region

`dio.setRegion (...)` procedure is invoked on 'dio' object and adds a 'Region' to the object. The procedure takes inputs similar to the `Region` object in config file as described in Subsec. 2.1.3. This procedure accepts the name of only one 'RefWin' as input.

If the given region is composed of a union of multiple 'RefWins', then `setCompositeRegion (...)` procedure can be invoked to add such a region.

Alternately, `dio.addRegion(...)` procedure can be invoked to create a region from an external `RefWin` object.

3.1.4 Defining Doping Profile

A doping profile can be added to the 'dio' object using `dio.setDopingDef (...)` function. It takes inputs similar to the `DopingDef` object in config file as explained in Subsec. 2.1.4. Additional input parameters in this function are described in the next chapters.

Alternately, `dio.addRegion(...)` procedure can be invoked to create a region from an external `RefWin` object.

3.1.5 Defining Contacts

A contact is added to the 'dio' object by using `dio.setContact (...)` function. It takes two inputs, namely, the name of the contact (`Name`) and the 'RefWin' associated with it (`RefWin`). If a contact already exists with the given name, the RefWin is added to the existing contact, instead of creating a new one.

Alternately, `dio.addContact(...)` procedure can be invoked to create a contact from an external `RefWin` object.

3.1.6 Defining Meshing rules

The procedure `dio.setMeshDef (...)` adds a new meshing rule to the existing meshing rules of 'dio'. This new meshing rule is applied to the volume confined only in the RefWin named `RefWin`. The procedure takes input parameters similar to those defined in Subsec. 2.1.6. The parameters which define the grid spacing along x, y, and z axis are set using `Xarg`, `Yarg`, and `Zarg` as a list of four numbers as follows `Xarg=[0., 0.0004, 0.04, 1.15]` The four numbers are `cen`, `minspacing`, `maxspacing`, `incr` in the same order.

Alternately, `dio.addMeshDef(...)` procedure can be invoked to create a new meshing rule which is valid inside an external `RefWin` object.

3.2 Generating a device

In this section, the above python script is used to generate the pn-diode structure shown in Fig. 2.1. The structure is created using the command

```
>> python diode_str.py.
```

The above script will generate '<Name>_str.h5' file together with a script file '<Name>_str.xdmf'. If Paraview is installed on the machine, the xdmf file can be opened using the following command

```
>> paraview diode_str.xdmf
```

The above command will open `paraview`. The same steps as described in Sec. 2.2 can be followed. Generated mesh and doping profile will look exactly as shown in Fig. 2.1(a) and in Fig. 2.1(b),

respectively. Contact vertices set by the python file also look as shown in Fig. 2.1(c).

3.3 Modifying the device

Copy the above python file to any other file and open it in any text editor. Modify the mesh definitions as follows.

```
dio.setMeshDef (Name="MDefSi", RefWin="RefSi",
    Xarg=[0.4, 1E-4, 0.05, 1.3], Yarg=[0., 0.025, 0.025, 1.],
    Zarg=[0., 0.025, 0.025, 1.])
```

Perform meshing again using the following command.

```
>> python diode_str.py
```

Using the modified settings will generate mesh exactly as shown in Fig 2.2(a). Now, modify doping of the device by changing the doping RefWin as follows.

```
dio.setRefWin (Name="RefPDoping", Shape="Rectangle",
    Position=np.array ([[xmin, ymin, 0.], [xmid+0.4, ymax, 0.]]))

dio.setRefWin (Name="RefNDoping", Shape="Rectangle",
    Position=np.array ([[xmid+0.4, ymin, 0.], [xmax, ymax, 0.]]))
```

Rerun meshing using the above command. View the doping concentration. Now p-n junction is shifted to the right by 0.4μm as shown in Fig. 2.2(b). In this way, the meshing or doping can be modified using python interface.

Chapter 4

Configuring RefWin Object

A RefWin in structure generator can be defined in a config file using the following script.

```
RefWin*RefSi:
{
Position: ([-1.0, -0.2, 0.], [1., 0.2, 0.]);
Shape = "Rectangle";
}
```

Name of the RefWin is specified by the string after 'RefWin*'. `Position` argument takes a number of points as inputs each of which is specified by an array of *three* floating points. The number of points to be specified and their meaning depends on the `Shape` of the RefWin. Various 'shapes' which are recognized by the structure generator are listed below.

4.1 1D segment

A segment is set by providing two points in the `Position` argument and setting `Segment` as shape. It can be axis aligned or slanted. In a

2D device, care must be taken to set z coordinate of both the points to 0.

4.2 2D shapes

Note, that in a 2D device all the RefWins must be in XY plane only. 2D devices in YZ or XZ coordinates are not supported at the moment.

4.2.1 Rectangle

An axis-aligned `Rectangle` is set by specifying the two endpoints of any one of its diagonals in `Position` argument. These two points must have either x, y, or z coordinate same. The rectangle can lie in XY, YZ, or XZ planes, depending on whether *all* the points have the same z-coordinate (XY plane), x-coordinate (YZ plane), or y-coordinate (XZ plane).

4.2.2 Polygon

A `Polygon` shape is be defined by listing point coordinates of all the vertices of the polygon in counterclockwise or clockwise order in `Position` argument. All the points of the polygon must be co-planar, else the program outputs error and exits.

4.2.3 Circle

A `Circle` shape is set by specifying its center point in `Position` argument and passing its radius as an additional argument as follows: `Radius=0.5`. The circle object can only be defined for a 2D device. Therefore, it can only lie in XY plane.

4.2.4 Ellipse

An `Ellipse` shape is set by specifying the point of intersection of short and long diameter as the first point in `Position` argument. Also, one of the endpoints of the long diameter is set as second point in `Position`. Short radius is set by `Radius` argument. The ellipse object

can only be defined for a 2D device. Therefore, it can only lie in XY plane.

4.3 3D shapes

4.3.1 Cuboid

An axis-aligned `Cuboid` is set by specifying the two endpoints of any one of its diagonals in `Position` argument. These two points must have different x-,y-, or z- coordinates, else the program gives an error and exits.

4.3.2 Sphere

A `Sphere` shape is set by specifying its center point in `Position` argument. Its radius is set by `Radius` argument.

4.3.3 Ellipsoid

An `Ellipsoid` shape is set by specifying the point of intersection of short and long diameter as the first point in `Position` argument. Also, one of the endpoints of the long diameter is set as second point in `Position`. Short radius is set by `Radius` argument. Currently, only a protruding ellipse can be specified.

4.3.4 Cylinder or Cone

To set `Cylinder` or a `Cone`, arguments of the elliptical shape (or circular shape) at the base of these shapes must be set. They are set by specifying the point of intersection of short and long diameter of the base ellipse as the first point in `Position` argument and one of the end points of the long diameter as second point in `Position`. Short radius of the base ellipse is set by `Radius` argument. If the short radius is not set or is equal to the long radius (distance between first and second point in `Position`), then the base is circular. Point at the top of the cone or the center-point at the top of the cylinder is specified as the third point in `Position`.

4.3.5 Convex hull

A polyhedron which is a 'convex hull' of all the points specified in `Position` argument is created by setting the shape as `ConvexHull`. All the points provided in `Position` must not be co-planar.

4.3.6 Polyhedron

A `Polyhedron` is a special shape, as it does not accept `Position` as an argument. Instead, it is set by specifying all of its faces as a list of 2D RefWins. This means, that all the faces of the polyhedron must be defined as 2D RefWins beforehand. Names of the RefWins which form its faces are provided as a comma separated list to the `SurfaceRefWin` argument.

4.4 Python Interface

4.4.1 Position object

`mesher.position(...)` constructor creates an instance of a position, when x, y, z coordinates are supplied. A list of points specified by the list of `positions` is input as an argument while creating a `RefWin` in python interface. Following mathematical operations can be directly performed on the `position` objects.

- `X()`,`Y()`,`Z()`: returns x,y,z coordinates of the point.
- `Binary operations` +,-,*,/ : Binary operations are performed component-wise
- `Assignment operations` =+,=-,=*,=/ : In-place component-wise binary operations.
- `(in)equality` ==, != : Two points are equal if *each* of the x,y,z coordinates of the two points are less than 10^{-5} away.
- `Scalar operations` *, *=, /, /= : Operation of a point with a scalar.

Note: The position object can also be used to specify "direction" vector.

4.4.2 Add RefWin to the device

`setRefWin (...)` procedure is invoked on the device object adds a 'RefWin' to the device. Inputs are similar to those listed above. Note, that `Position` argument takes coordinates in the form of a list of `position` instances. The leading dimension has the size of number of points and the trailing dimension has the size of 3, corresponding to X, Y, and Z coordinates of the points. For example, a rectangular RefWin is added to the device object 'dio' using the following command in python script.

```
dio.setRefWin (Name="RefSi", Shape="Rectangle",
    Position=[m.position(-1.,-1.,0.), m.position(1.,1.,0.)],
          NumericParams={},NumericListParams={})
```

Also, the name of the RefWin is unique to it. Adding multiple RefWins with the same `Name` gives an error message and the RefWin is not added.

4.4.3 Create RefWin Object

`mesher.refwin(...)` constructor creates a new `RefWin` when the following arguments are provided.

- `Device`: Device object (e.g. `dio`).
- `Name`: Name of the RefWin.
- `Shape`: RefWin shape.
- `Position`: A list of points as a list of `positions`.
- `NumericParams`: Python dictionary mapping parameter names and their values in number.
- `NumericListParams`: Python dictionary mapping parameter names and their values in a list of numbers.
- `BaseNormalAxis`: Axis normal to the base. Accepts 'X', 'Y', or 'Z' as arguments. Default 'Y'.

Various operations can be performed on the RefWin object(s). They are described in the next section.

4.5 Modifying RefWin object

An object of a RefWin is created by `win = mesher.refwin(...)`. Following operations can be applied on the object.

4.5.1 General

getDim

`win.getDim()` procedure returns dimension of the shape.

getName

`win.getName()` procedure returns name of the RefWin.

setName

`win.setName("Name")` procedure sets name of the RefWin.

isEmpty

`win.isEmpty()` procedure returns if the RefWin is empty or not.

4.5.2 Shape Transformation

Translation

`win.translateBy(position(x,y,z))` procedure translates the RefWin shape in 2D/3D space by `position(x,y,z)`. The translation vector is provided as a `position` object.

$$\vec{p}_{new} = \vec{p} + \vec{c} \tag{4.1}$$

Here, $\vec{p}$ and $\vec{p}_{new}$ are the current and new positions of the vertex. $\vec{c}$ is the user-given translation vector.

Scaling

`win.scaleBy(val, position(x,y,z))` procedure scales the RefWin shape in 2D/3D space by a factor `val`. Scaling is performed relative to the point given by `position(x,y,z)` object.

$$\vec{p}_{new} = s \cdot (\vec{p} - \vec{c}) \tag{4.2}$$

Here, $\vec{p}$ and $\vec{p}_{new}$ are the current and new positions of the vertex. $\vec{c}$ is the user-given point, and s is the scaling factor.

Rotation from a point to another

`win.rotateFromTo(position(x1,y1,z1),position(x2,y2,z2))` procedure rotates the RefWin shape in 2D/3D space by the shortest route from `position(x1,y1,z1)` to `position(x2,y2,z2)` about the shape center.

Rotation around an axis

`win.rotateAroundAxis(angle,position(x1,y1,z1))` procedure rotates the RefWin shape in 2D/3D space by `angle` degrees about an axis along the direction `position(x1,y1,z1)`.

Rotation around an axis at point

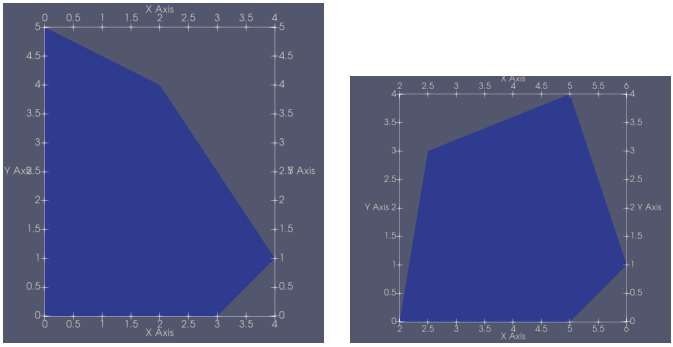
`win.rotateAroundAxisAtPoint(angle,position(x1,y1,z1),position(x2,y2,z2` procedure rotates the RefWin shape in 2D/3D space by `angle` degrees about an axis specified by `position(x1,y1,z1)` sp.

Shape mirroring

`win.mirrorByPlane(position(x1,y1,z1))` procedure mirrors the RefWin shape in 2D/3D space. Mirror-plane is defined by the plane-normal `position(x1,y1,z1)` passing through the shape center.

4.5.3 Boolean operation

Various boolean operations can be performed on a pair of `RefWin` objects. Here, these operations are applied on a RefWins `win1` and



(a) shape of "win1" (b) shape of "win2"

Figure 4.1: Shapes of the RefWins (a) "win1" and (b) "win2".

`win2` which are created by a python script given below. The original RefWins are shown in Fig. 4.1(a) and Fig. 4.1(b).

```
import mesher as tm
poly = [tm.position(0., 0.,0.),tm.position(3.,0.,0.),
        tm.position(4.,1.,0.),tm.position(2.,4.,0.),
        tm.position(0.,5.,0.)]
win1 = tm.refwin("RefCyl1","Polygon",poly,{},{})

poly2 = [tm.position(2., 0.,0.),tm.position(5.,0.,0.),
         tm.position(6.,1.,0.),tm.position(5.,4.,0.),
         tm.position(2.5,3.,0.)]
win2 = tm.refwin("RefCyl2","Polygon",poly2,{},{})
```

Add

`win.add(refwin=win2)` procedure creates a union of the geometric shapes of `win2` and `win`, and assigns the new shape to `win2`.

```
win1.add(refwin=win2)
```

RefWin "win1" modified by the above line is shown in Fig. 4.2.

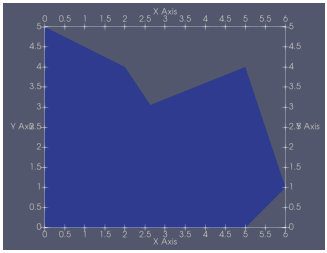


Figure 4.2: Shape of RefWin "win1" after adding "win2" to it.

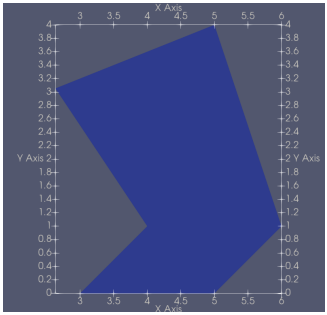


Figure 4.3: Shape of RefWin "win2" after subtracting "win1" from it.

Subtract

`win.subtract(refwin=win2)` procedure subtracts the geometric shape of `win2` from the geometric shape of `win`, and assigns the new shape to `win2`. The new shape contains all the points which are present in `win` but not in `win2`.

```
win2.subtract(refwin=win1)
```

RefWin "win2" modified by the above line is shown in Fig. 4.3.

Clip

`win.clip(refwin=win2)` procedure creates an intersection of the geometric shape of `win2` from the geometric shape of `win`, and assigns

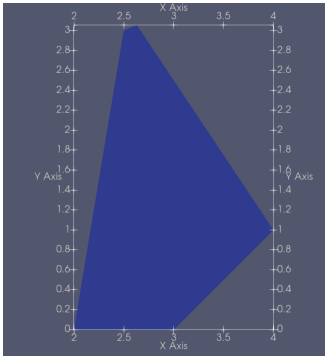


Figure 4.4: Shape of RefWin "win1" after clipping with "win2".

the new shape to `win2`. The new shape contains all the points which are present in both `win2` and `win`.

```
win1.clip(refwin=win2)
```

RefWin "win1" modified by the above line is shown in Fig. 4.4.

overlaps

`win.overlaps(refwin=win2)` procedure returns true if the geometric shape of `win2` overlaps with that of `win`.

> **Note:** Boolean operations can be performed on the two RefWin objects of the same dimension (2D/3D). Performing boolean operation on a 3D RefWin with a 2D RefWin and vice-versa may lead to an undefined shape.

4.5.4 2D/3D shape editing

Various procedures useful for modifying a 2D/3D `RefWin` are listed below. They are applied on the RefWin "win1" shown in Fig. 4.1(a).

Chamfering

`win.chamferAt([position(x1,y1,z1),...],[r1,...],True/False)` procedure chamfers the edges of the RefWin on which the points

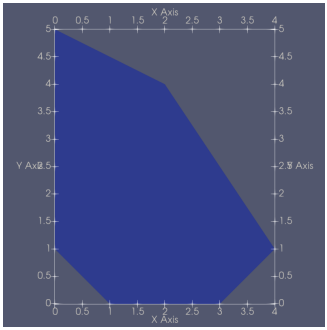


Figure 4.5: Shape of RefWin "win1" after applying chamfering operation.

specified by position list lie. For each of the points, chamfer radius is specified by a list of floats. If the point coincides with the shape corner, then all the edges connected to the corner are rounded. If the last argument is set to true, then all the edges of the shape are chamfered.

```
win1.chamferAt([tm.position(0., 0.,0.)],[1.0])
```

RefWin "win1" modified by the above line is shown in Fig. 4.5.

Rounding

`win.roundAt([position(x1,y1,z1),...],[r1,...],True/False)` procedure rounds the edges of the RefWin on which the points specified by position list lie. For each of the points, rounding radius is specified by a list of floats. If the point coincides with the shape corner, then all the edges connected to the corner are rounded. If the last argument is set to true, then all the edges of the shape are chamfered.

```
win1.roundAt([tm.position(0., 0.,0.),tm.position(3.,0.,0.)],[0.5,0.5]
```

RefWin "win1" modified by the above line is shown in Fig. 4.6.

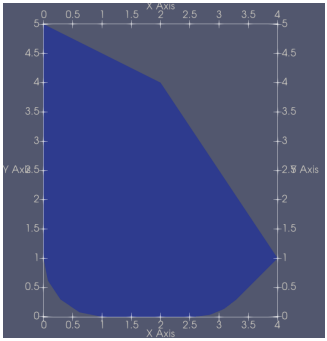


Figure 4.6: Shape of RefWin "win1" after applying rounding operation.

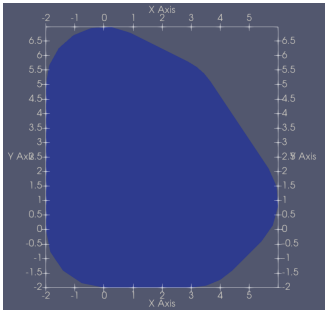


Figure 4.7: Shape of RefWin "win1" after applying offsetting operation.

Offsetting

`win.offsetRefWin(thickness)` modifies the existing 2D/3D shape by expanding/contracting it along the *normal* to its surface. Negative value of `thickness` results in contraction, whereas positive value expands the existing shape.

```
win1.offsetRefWin(2.0)
```

RefWin "win1" modified by the above line is shown in Fig. 4.7.

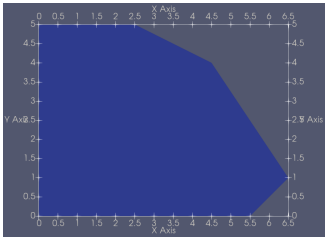


Figure 4.8: Shape of RefWin "win1" after stretching it along x-axis by 2.5μm.

Stretch

`win.stretchRefWin(position(x1,y1,z1))` stretches the existing 2D/3D shape along the direction specified by `position(x1,y1,z1)`. The stretch distance is equal to the norm of the specified `position` object. Stretching is performed such that the surface of the RefWin retains its shape.

```
win1.stretchRefWin(tm.position(2.5,0.,0.))
```

RefWin "win1" modified by the above line is shown in Fig. 4.8.

Stretch at a point

`win.stretchRefWinAtPoint(position(x1,y1,z1),position(x2,y2,z2))` stetches the cross-section of the existing 2D/3D shape at a point `position(x1,y1,z1)` in the plane perpendicular to the direction specified by `position(x2,y2,z2)`. Among the two parts of the structure on either side of the cut-plane, the structure on the same side as the stretch direction is shifted, whereas the one on the other side is kept the same. If the cut-plane does not cut the structure, and if the structure lies on the same side as the stretch direction, then the entire structure is shifted by the direction. Otherwise, the structure is not modified.

```
win1.stretchRefWin(tm.position(3.5,0.,0.),tm.position(2.5,0.,0.))
```

RefWin "win1" modified by the above line is shown in Fig. 4.9.

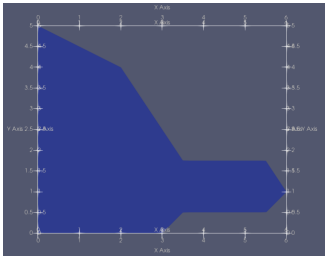


Figure 4.9: Shape of RefWin "win1" after stretching it at a point (3.5,0,0) along x-axis by 2.5μm.

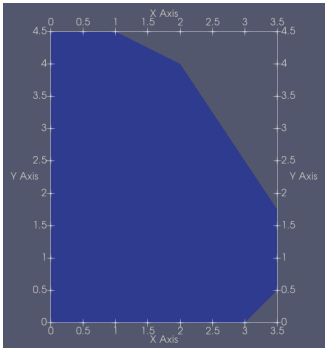


Figure 4.10: Shape of RefWin "win1" after limiting its span to $0 < x < 3.5$ and $0 < y < 4.5$.

Limit Span

`win.limitRefWinSpan(position(x1,y1,z1),position(x2,y2,z2))` procedure clips the shape to a box defined the two diagonally opposite corners given by `position(x1,y1,z1)` and `position(x2,y2,z2)`.

```
win1.limitRefWinSpan(tm.position(0., 0.,0.),tm.position(3.5, 4.5,0.))
```

RefWin "win1" modified by the above line is shown in Fig. 4.10.

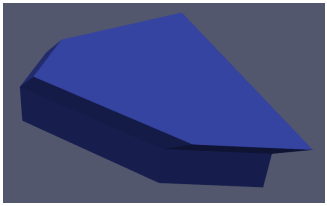


Figure 4.11: Shape of RefWin "win1" after sweeping it along a path defined by the points - $(0,0,0), (0,0,1), (0,1,2)$

4.5.5 2D to 3D shape conversion

Following operations convert a 2D RefWin to a 3D RefWin.

Sweep along a path

`win.sweep2DRefWin([position(x1,y1,z1),...])` takes a list of points as position objects as an argument, and creates a 3D shape by sweeping the 2D shape of the RefWin along the piecewise-linear path formed by the list of points. Direction of each of the segment of the piecewise-linear path *must not be* parallel to the 2D shape. Such a segment is ignored.

```
win1.sweep2DRefWin([tm.position(0.,0.,0.),
      tm.position(0.,0.,1.), tm.position(0.,1.,2.)])
```

RefWin "win1" modified by the above line is shown in Fig. 4.11.

Sweep along a direction

`win.makePrism2DRefWin(position(x,y,z))` sweeps the 2D shape along the direction specified by `position(x,y,z)`. Length of the sweep is equal to the norm of the direction. No shape is generated if the specified direction is parallel to the plane of the 2D shape.

```
win1.makePrism2DRefWin(tm.position(0.,0.,5.))
```

RefWin "win1" modified by the above line is shown in Fig. 4.12.

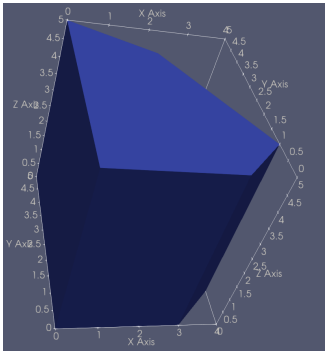


Figure 4.12: Shape of RefWin "win1" after sweeping it by 5μm along z-direction.

Make pyramid

`win.makePyramid2DRefWin(position(x,y,z))` creates a pyramid which has the 2D shape as a base and the point specified by `position(x,y,z)` as an "apex". The program terminates giving error if the apex point is in the same plane as the RefWin.

```
win1.makePyramid2DRefWin(tm.position(1.,1.,5.))
```

RefWin "win1" modified by the above line is shown in Fig. 4.13.

Revolve around axis

`win.revolve2DRefWin(position(x,y,z),position(dx,dy,dz),angle)` revolves the 2D shape around the axis passing through the point given by `position(x,y,z)` and has the direction given by `position(dx,dy,dz)`. The 2D shape is revolved by `angle` degrees around the axis to create the 3D shape. The axis must be in the same plane as that of the 2D shape and the shape must lie entirely on one side of the axis.

```
win1.revolve2DRefWin(tm.position(0.,0.,0.),
                tm.position(0.,1.,0.), 360.)
```

RefWin "win1" modified by the above line is shown in Fig. 4.15.

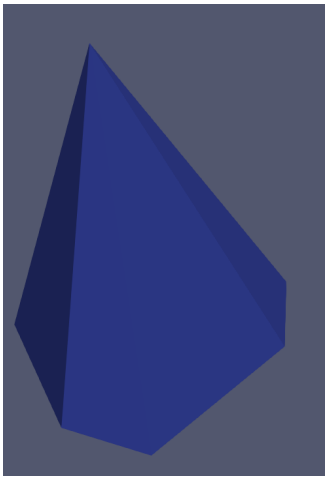
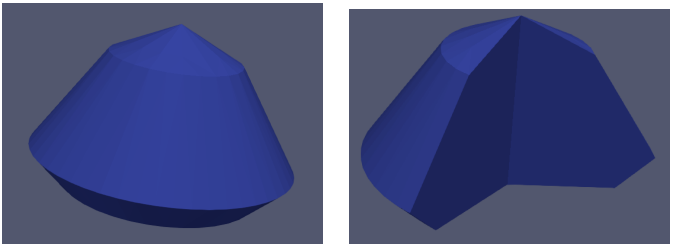


Figure 4.13: Shape of RefWin ”win1” after creating a pyramid out of it.



(a) Revolving “win1” by 360° (b) Revolving “win1” by 230°

Figure 4.14: Shape of RefWin ”win1” after revolving it by (a) 360°and (b) 230°.

4.5.6 Trapezoidal 3D etch

Following commands can create a custom etch-profile in 3D when applied to a 2D mask RefWin in the XZ plane. Note, that creation of a custom etch-profile in 2D can be achieved by defining a polygonal RefWin with the etch profile.

```
import mesher as tm
winSi = tm.refwin("RefSi","Rectangle", [tm.position(-0.25, 0., -0.25)
winSi.roundAt([tm.position(-0.25, 0., -0.25)],[0.1])
```

From depth-angle list

`win.sweep2DMaskDepthAngle([dy1,ang1,dy2,ang2,...])` creates a piecewise trapezoidal 3D shape from a 2D RefWin in XZ plane. The shape is created by applying successive trapezoid etching using the RefWin as a mask. The function takes a list of numbers as input. Numbers at odd positions in the list (e.g. `dy1`, `dy2`, ...) specify etch depth (in μm) of the successive trapezoidal etches. Numbers at even positions in the list (e.g. `ang1`, `ang2`, ...) specify angle in degrees of the etch side-walls with the etch-direction in each of the successive trapezoidal etches. Positive angle implies outward slant of the side-wall.

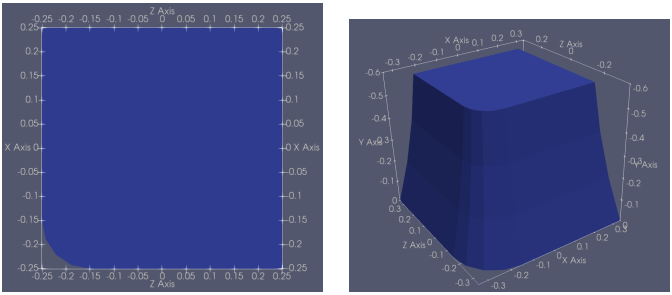
```
winSi.sweep2DMaskDepthAngle([-0.2,5.71,-0.2,8.53,-0.2,11.3])
```

Here, -0.2 is depth. Negative value indicates shape "growth" in negative direction. 5.71°is angle in degrees of the sidewalls with the growth direction. Changing the sign of the angle would bend the sidewalls "inward".

RefWin "winSi" modified by the above line is shown in Fig. **??**.

From etch profile

`win.sweep2DMaskPositionList([position(x1,y1,z1),position(x2,y2,z2)...` creates a piecewise trapezoidal 3D shape from a 2D RefWin in XZ plane. The shape is created by applying successive trapezoid etching using the RefWin as a mask. The function takes a list of positions as input. Difference between consecutive points in the list is converted to the pairs of etch-depth and etch-angle as follows. Etch-depth is



(a) Etch mask (b) Trapezoidal etch shape

Figure 4.15: (a) Shape of RefWin 'mask' in XZ plane and (b) Shape of RefWin after using the mask for trapezoidal etch.

given by the y-component of the difference $(y2 - y1)$. Corresponding etch-angle is calculated as $\arctan(\frac{\sqrt{dx^2+dz^2}}{dy})$.

```
winSi.sweep2DMaskPositionList([tm.position(0,0,0),
tm.position(-0.02,-0.2,0.),tm.position(-0.05,-0.4,0.),
tm.position(-0.09,-0.6,0.)])
```

The shape obtained by the above line is identical to the shape obtained by the python line in the previous sub-subsection.

4.5.7 Saving the shape

Save STL file

`win.saveSTL()` saves the shape of the RefWin in an STL file - <Name>.stl. Here, <Name> is the RefWin name which can be set by `win.setName(name)`.

Chapter 5

Configuring Regions, Contacts, and Doping

A semiconductor device or an optoelectronic wave-guide is composed of a number of objects such as regions, contacts, and doping profiles. They are created by associating material properties with the volumes encompassed by the 'RefWins'.

5.1 Region objects

A `Region` is created by associating material properties with one or more 'RefWins'. It can be instantiated in a config file as follows,

```
Region*RegLeft:
{
  RefWin = ["RefLeft"];
  Material = "Silicon";
  DuplicateInterfaceVertices = ["RegMiddle"];
  MaxAreaBulk = 0.1;
}
```

Region name is specified by a string after 'Region*' string. The above region is named 'RegLeft'. A comma separated list of names of the

RefWins which constitute the region are passed to the region object by `RefWin` keyword. Material of the region is set by `Material` keyword.

If 'TetMesh' is specified as a meshing type, then the mesh refinement is determined by the following keywords.

- `MaxAreaBulk` : Maximum area/volume of the triangle/tetrahedral elements in the bulk of the region.
- `MaxAreaInterface` : Maximum area/volume of the triangle/tetrahedral elements at the interface of the region.
- `MinDopingGradient`, `MaxDopingGradient`, `AreaMinGradient`, and `AreaMaxGradient` : Parameters which define doping-dependent mesh refinement.

For certain hetero-junction simulations, vertex at the interface between two regions need to be duplicated such that one vertex belongs to first region and the second vertex to second region. The region names with which the given region has duplicated interface vertices are specified as a comma separated list by `DuplicateInterfaceVertices` keyword.

5.1.1 Python script interface

A region can be added to the device object in the python script using the procedure `setRegion` on the 'device' as follows.

```
dio.setRegion (Name="RegSi", RefWinName="RefSi",
    Material="Silicon", MeshingParams={"MaxAreaBulk" : 0.1})
```

Here, 'dio' is the device object instantiated in python script file. Other arguments, such as `MaxEdgeLength` can be specified as a python dictionary. Note that this procedure takes only a single RefWin as input. If a region of a union of multiple RefWins is to be set, use the following command,

```
dio.setRegion (Name="RegSi", RefWinNames=["RefSi1", "RefSi2"],
    Material="Silicon")
```

The above command takes names of multiple RefWins as input parameters.

A region can also be added by specifying a `RefWin` object created by its Python constructor as follows.

```
win = m.refwin("RefSi2","Rectangle",
               [m.position(1, 0, 0.),
                m.position(0, 1, 0.)],{},{})
dio.addRegion (Name="RegSi2", refwin=win,
               Material="Silicon", MeshingParams={})
```

Note, that the procedure `dio.addRegion(...)` is used above to input the `RefWin` object `win`.

5.2 Contact objects

A `Contact` is composed of a number of vertices which are located in the volume encompassed by the RefWins specified as a list of comma separated strings with the keyword `RefWin`. It can be instantiated in a config file as follows,

```
Contact*Cathode:
{
  RefWin = ["RefCathode"]
}
```

The string after 'Contact*' is the name of the contact. The above contact is named 'Cathode'. It is an *electrical contact* in which all the vertices of the contact have a fixed potential.

5.2.1 Thermal Contact

A `ThermalContact` is specified in the same way as an electrical `Contact` as follows,

```
ThermalContact*Sink: {
  RefWin = ["RefCathode"]
}
```

The above contact is named 'Sink'. It is a *thermal contact* in which all the vertices of the contact have a fixed temperature in coupled electro-thermal simulations.

5.2.2 Python script interface

An electrical contact can be added to the device object using the following python procedure.

```
dio.setContact (Name="Cathode", RefWinName="RefCathode")
```

Note, that the above procedure accepts only one RefWin as an input. Multiple RefWins can be linked to the same contact by calling the above procedure with different RefWin names, repeatedly.

Similarly, a thermal contact can be added to the device object using the procedure `setThermalContact`.

A contact can also be added by specifying a `RefWin` object created by its Python constructor as follows.

```
dio.addContact (Name="RegSi2", refwin=win)
```

Note, that the procedure `dio.addContact(...)` is used to input the `RefWin` object `win`.

5.3 Doping profiles

Dopant distribution in a semiconductor device is set by defining the doping profiles. Each doping profile is active in a particular RefWin specified by the keyword `RefWin`. The doping profile gives concentration of the dopant species specified by `Dopant` at each of the vertices in the RefWin. Total concentration of each dopant species at each vertex is obtained by *adding* contributions from each of the doping profiles at that vertex.

Note, that calculation of dopant concentration at each vertex is performed only after meshing the structure (although dopant profiles can be defined before).

A dopant profile shape is specified by `Type`. The following profile shapes can be specified.

- `Constant`
- `Gaussian`
- `Exponential`
- `Linear`
- `Analytic`

5.3.1 Constant doping profile

When the profile type is set to `Constant`, a constant concentration given by `Concentration` of a dopant set by `Dopant` is added to all the vertices in the RefWin whose name is specified by `RefWin`. An example of constant profile is given below.

```
DopingDef*Ndoping:
{
RefWin = "RefNDoping"
Type = "Constant"
Concentration = 1E18
Dopant = "Phosphorus"
}
```

5.3.2 Gaussian doping profile

Gaussian dopant distribution is created by specifying a base RefWin by the keyword `BaseWin` together with the `RefWin`. In 3D devices, the `BaseWin` is a 2D rectangle or polygon, whereas in 2D devices, the `BaseWin` is a 1D segment. The peak concentration (C) of the profile is set by `Concentration` and standard deviation (σ) is set by `DecayLength`. Depth (μ) of the Gaussian peak from the BaseWin is set by `Depth`. Gaussian peak in one or the other half-space of the BaseWin plane can be chosen by changing the sign of `Depth`.

At each vertex at location $\vec{r}$ lying inside the `RefWin`, perpendicular distance ($d_\perp$) to the `BaseWin` is calculated together with lateral distance ($d_\parallel$) from the edges of the BaseWin polygon in 3D or from endpoints of the BaseWin in 2D. If the projection of the point onto the

BaseWin lies inside the BaseWin polygon, then $d_{\parallel} = 0$. Thus, $d_{\parallel} \geq 0$. Doping concentration at each vertex is given by,

$$C_i(\vec{r}) = C \cdot \text{erf}\left(-(\frac{d_{\parallel}}{\sigma})^2\right) \cdot \exp\left(-\frac{1}{2}(\frac{d_{\perp} - \mu}{\sigma})^2\right) \tag{5.1}$$

Here, i is the doping profile id. Total doping at a given vertex is calculated by adding up dopant concentration at the given vertex from all such profiles. An example of a Gaussian profile is given below.

```
DopingDef*Pdoping:
{
RefWin = "RefPDoping";
Type =  "Gaussian";
Concentration = 5E16;
Depth = -0.1;
DecayLength = 0.1;
BaseWin = "PBaseLine";
Dopant = "Boron";
}
```

5.3.3 Exponential doping profile

Exponential dopant distribution takes the same set of arguments as Gaussian profile explained in Subsec. 5.3.2. It differs from Gaussian profile in the formula to calculate dopant concentration at each vertex at location $\vec{r}$ in the `RefWin`. The formula is given by,

$$C_i(\vec{r}) = C \cdot \exp\left(-\frac{d_{\parallel}}{\sigma}\right) \cdot \exp\left(-\frac{|d_{\perp} - \mu|}{\sigma}\right) \tag{5.2}$$

Here, C is set by `Concentration`, σ is set by `DecayLength`, μ is set by `Depth`, $d_{\perp}$ is perpendicular distance from the `BaseWin` and $d_{\parallel}$ is the distance of projection of the vertex onto the BaseWin plane from the BaseWin edge.

5.3.4 Linear doping profile

Linear dopant profile takes the same set of arguments as Gaussian profile explained in Subsec. 5.3.2. It differs from Gaussian profile in the

formula to calculate dopant concentration at each vertex at location $\vec{r}$ in the `RefWin`. The formula is given by,

$$C_i(\vec{r}) = C \cdot \max\left(1 - \frac{d_{\parallel}}{\sigma}, 0\right) \cdot \max\left(1 - \frac{|d_{\perp} - \mu|}{\sigma}, 0\right) \tag{5.3}$$

Here, C is set by `Concentration`, σ is set by `DecayLength`, μ is set by `Depth`, $d_{\perp}$ is perpendicular distance from the `BaseWin` and $d_{\parallel}$ is the distance of projection of the vertex onto the BaseWin plane from the BaseWin edge.

5.3.5 Analytic doping profile

When `Analytic` profile is specified, a user-defined expression is used to calculate doping concentration at each vertex inside the `RefWin`. The expression is provided as a string of numbers, common functions, and mathematical operations with `AnalyticExpr` keyword. In the expression, X, Y, and Z are the symbols reserved for x, y, and z coordinate at each vertex. Thus, doping concentration at a vertex at location $\vec{r} = X\hat{x} + Y\hat{y} + Z\hat{z}$ is given by,

$$C_i(\vec{r}) = C \cdot f(X, Y, Z) \tag{5.4}$$

Here, C is set by `Concentration`. An example of a user-defined analytic profile is given below.

```
DopingDef*Pdoping:
{
RefWin = "RefPDoping";
Type =  "Analytic";
Concentration = 5E16;
AnalyticExpr = "abs(X*Y)+0.4*exp(-X*X)";
Dopant = "Boron";
}
```

Here, the user-defined function to define the doping profile is,

$$f(x, y, z) = |x \times y| + 0.4 \times \exp\left(-x^2\right) \tag{5.5}$$

In this way, various doping profiles can be created from the user-defined expressions. The structure generator uses 'exprtk' package for parsing

the expression. All the mathematical operators and common functions which are defined in 'exprtk' function can be used in the user-defined expression.

5.3.6 Python script interface

A doping profile can be added to the device object 'dio' using the command `setDopingDef`. For example, a constant doping profile can be added as follows.

```
dio.setDopingDef (Name="Pdoping", RefWinName="RefPDoping",
    Type="Constant", Concentration=5E16,
    DopantSpecies ="Boron")
```

Similarly other types of doping profiles can be added by specifying additional keywords such as `BaseWin`, `DecayLen`, `Depth`, `AnalyticExpr`, etc.

A doping profile can also be added by specifying a `RefWin` object created by its Python constructor as follows.

```
dio.addDopingDef (Name="Pdoping", refwin=win,
    Type="Constant", Concentration=5E16,
    DopantSpecies ="Boron")
```

Note, that the procedure `dio.addDopingDef(...)` is used to input the `RefWin` object `win`.

5.4 Editing device

The following commands edit the entire device. These commands can only be called on a python device object `dio`.

5.4.1 Mirror device

An example command is given by –

```
dio.mirrorDevice(Axis="X",KeepOriginal=True,MergeRegions=True)
```

The command reflects the device structure along with the mesh along the axis specified by `Axis`. Reflection takes place at the extremum (max/min) coordinate. For example, if Axis="X", the device is

reflected on the positive X axis at the maximum x-coordinate. If Axis="-X", the device is reflected on the negative X axis at the minimum x-coordinate. If the argument `KeepOriginal` is set true, the mirrored device is merged to the original device. If `MergeRegions` is true, regions of the mirrored device are merged to the original device.

5.4.2 Translate device

An example command is given by –
`dio.translateDevice(tm.position(x1,y1,z1))` The command translates the device structure along with the mesh by `tm.position(x1,y1,z1)`.

5.4.3 Stretch device

An example command is given by –
`dio.stretchDeviceAtPoint(tm.position(x1,y1,z1),tm.position(x2,y2,z2))` The command stretches the device structure (i.e. all the regions, doping definitions, and contacts) at a point `tm.position(x1,y1,z1)` along the direction `tm.position(x2,y2,z2)`. The device is remeshed.

5.5 Save and Load

Various save and load functions have been provided to save various entities (regions, mesh, contact, and doping definitions) created in the device definition. Definitions of these entities are stored in binary files (*.bin). They can be loaded to another device thereby allowing copy-paste of these entities. These functions are described below.

5.5.1 Structure - save and load STEP

An example command to save the device structure as a STEP file is given by –
`dio.saveStructureAsSTEP("file_bdr.step")` The command saves all the regions in a step file. The region names as well as materials are also stored. This step file can be viewed in any step file viewer. A recommended viewer is `cadassistant` by OpenCascade.

The stored STEP file can be loaded to another device object by using the following command –

`dio2.loadStructureFromSTEP("file_bdr.step")` It loads the shapes from the STEP file to the device `dio2` and create a new region with the same material corresponding to each loaded shape.

5.5.2 Regions - save and load

An example command to save the listed regions as a binary file is given by –
`dio.saveRegionsAsBin("file_reg.bin", ["reg1","reg2"])` The command saves only the regions named `reg1` and `reg2` to the bin. If the list of region names is empty then all the regions are stored to the bin file.

The stored bin file containing regions can be loaded to another device object by using the following command –
`dio2.loadRegionsFromBin("file_reg.bin")` It loads the regions to the device `dio2` as new regions. Existing regions remain.

5.5.3 Doping Definitions - save and load

An example command to save the listed doping definitions as a binary file is given by –
`dio.saveDopingDefsAsBin("file_dop.bin", ["dop1","dop2"])` The command saves only the doping definitions named `dop1` and `dop2` to the bin. If the list of doping names is empty, then all the doping definitions are stored to the bin file.

The stored bin file containing doping definitions can be loaded to another device object by using the following command –
`dio2.loadDopingDefsFromBin("file_dop.bin")` It loads the doping definitions to the device `dio2` as new doping definitions. Existing definitions remain.

5.5.4 Mesh Definitions - save and load

An example command to save the listed mesh definitions as a binary file is given by –
`dio.saveMeshDefsAsBin("file_mdef.bin", ["mdef1","mdef2"])` The command saves only the mesh definitions named `mdef1` and `mdef2` to

the bin. If the list of mdef names is empty, then all the mesh definitions are stored to the bin file.

The stored bin file containing mesh definitions can be loaded to another device object by using the following command – `dio2.loadMeshDefsFromBin("file_mdef.bin")` It loads the mesh definitions to the device `dio2` as new mesh definitions. Existing definitions remain.

5.5.5 Contact Definitions - save and load

An example command to save the listed contacts as a binary file is given by –
`dio.saveContactDefsAsBin("file_cont.bin", ["cont1","cont2"])`
The command saves only the mesh definitions named `cont1` and `cont2` to the bin. If the list of contact names is empty, then all the contacts are stored to the bin file.

The stored bin file containing contacts can be loaded to another device object by using the following command – `dio2.loadContactDefsFromBin("file_mdef.bin")` It loads the mesh definitions to the device `dio2` as new mesh definitions. Existing definitions remain.

5.5.6 RefWins - save and load

An example command to save the listed RefWins as a binary file is given by –
`dio.saveRefWinsAsBin("file_ref.bin", ["ref1", "ref2"])` The command saves only the RefWins named `ref1` and `ref2` to the bin. If the list of RefWin names is empty, then all the RefWins are stored to the bin file.

The stored bin file containing RefWins can be loaded to another device object by using the following command – `dio2.loadContactDefsFromBin("file_ref.bin")` It loads the RefWins to the device `dio2` as new RefWins. Existing definitions remain.

Chapter 6

Configuring Mesh Generation

Regions defined by 'Region' objects are meshed to create finite element mesh or finite difference mesh of the structure which can be used for simulations. A quadtree/octtree based mesher (henceforth referred to as 'FEM-mesh') is provided natively in the meshing package for creating finite element mesh. Also, a rectangular/cubic element based mesher (referred to as 'tensor-mesh') is provided natively for creating finite difference mesh. Apart from that, various external mesh generators such as *Triangle* (for 2D meshing) and *TetGen* (for 3D meshing) are integrated into the meshing package, so that users can conveniently use them to generate mesh by setting specific keywords in the config file.

6.1 Mesh definition FEM or tensor-mesh

For FEM-mesh and the tensor-mesh generation, mesh-grid spacing along x, y, and z axes in a specific volume of the device can be specified by 'MeshDef' object, as follows.

```
MeshDef*MDefSiTop:
{
```

```
  RefWin = "RefAll";
  Xarg: [ 0., 0.0001, 0.05, 1.3 ]
  Yarg: [0., 0.025, 0.025, 1. ]
  Zarg: [0., 0.025, 0.025, 1. ]
}
```

The above `MeshDef` object sets parameters of the mesh generator program specific to the volume confined in the RefWin named `RefWin`. It also specifies the parameters which are used in defining grid along x, y, and z axis using the keyword `X`, `Y`, and `Z` defined as follows.

Grid parameters

Each of the `X`, `Y`, and `Z` grid definitions contains minimum grid-point spacing set by `Xarg[1]`, maximum grid-point spacing set by `Xarg[2]`. The grid points along that axis have spacing between the these values. If `Xarg[3]` factor is more than 1.0, then minimum grid spacing is multiplied with `Xarg[3]` factor for every subsequent grid spacing starting from grid point at `Xarg[0]`.

In tensor-mesh, grid-points along x-axis for all $x \in [x_{min}, x_{max}]$ are set by the definition given by `Xarg: [..]`, where x_{min} and x_{max} are the smallest and larges x-coordinates of `RefWin`. Similarly, grid-points along y- and z- axis are set by the definitions given by `Yarg: [..]` and `Zarg: [..]`, respectively.

On the other hand, in FEM-mesh, these mesh definitions are active only in the volume of the device confined by the `RefWin`.

6.2 Mesh settings in Triangle or Tetrahedral mesh

Mesh settings for *Triangle* or *TetGen* mesh engines are set region-wise in `Region` object. The following mesh settings can be set in each region definition.

- `MaxAreaBulk` : Maximum area/volume of the triangle/tetrahedral elements in the bulk of the region.

- `MaxAreaInterface` : Maximum area/volume of the triangle/tetrahedral elements at the interface of the region.
- `MinDopingGradient`, `MaxDopingGradient`, `AreaMinGradient`, and `AreaMaxGradient` : Parameters which define doping-dependent mesh refinement.

This restriction on voronoi volume is incorporated in both the external mesh engines to create refined mesh in the given region.

Note: The parameters noted above are considered *only* in *Triangle* meshing of a 2D device.

6.3 Meshing Options

Any of the above specified four types of mesh can be selected during mesh generation in the `Device` section of the config file. A sample `Device` section in a config file is shown below.

```
Device:
{
Name = "Diode";
MeshType = "FEM";
Simulation = "DD";
}
```

The argument `MeshType` recognizes the following keywords.

- `FEM`: Mesher performs Quadtree and Octtree based FEM meshing on 2D and 3D devices, respectively.
- `Tensor`: Mesher performs rectangular and cubic tensor meshing on 2D and 3D devices, respectively.
- `TriMesh`: Mesher uses *Triangle* package for meshing 2D devices.

An additional argument `Simulation` is used to specify the user's intent for the structure generation. It recognizes the following keywords.

- `DD`: Quantities required for drift-diffusion simulations are calculated.

- `QT`: Quantities required for quantum transport simulations are calculated.
- `FDTD`, `BPM`, `MODE`: Quantities required for optical simulations are calculated.
- `PRO`: Quantities required for process simulations are calculated.

Examples of generated mesh using each of the above meshing alternatives are presented below.

Note: The parameters noted anywhere else in the config file are ignored in *NetGen* mesher. Currently, region-specific mesh parameters cannot be passed to *NetGen mesher*.

6.3.1 Tensor Meshing

The structure generation config file provided in Chapter 3 is used to generate a tensor mesh, by setting `Tensor` as `MeshType`. Resulting mesh is shown in Fig. 6.1(c).

6.3.2 FEM Meshing

The structure generation config file provided in Chapter 3 is used to generate a FEM-mesh, by setting `FEM` as `MeshType`. Resulting mesh is shown in Fig. 6.1(a) .

6.3.3 Triangle Mesh Engine

The structure generation config file provided in Chapter 3 is used to generate a mesh using *Triangle* package, by setting `TetMesh` as `MeshType`. Resulting mesh is shown in Fig. 6.1(b).

6.3.4 NetGen Mesh Engine

The structure generation config file provided in Chapter 3 is modified to create a 3D Silicon device by adding depth to 'SiReg' region. It is then used to generate a mesh using *NetGen*, by setting `TetMesh` as `MeshType`. *NetGen* mesher settings are specified in `NetgenParams` section in the config file. The following keywords take a number as an argument.

- `maxh` : Maximum global mesh size allowed.
- `grading` : Mesh grading quality - 0: uniform, 1: local grading enabled.
- `fineness` : Mesh refinement quality - 0: coarse, 1: refined.
- `maxiter_refine` : Maximum number of iterations in 'iterative refinement'.
- `optsteps_3d` : Number of optimization runs for 3d meshing.

Following keywords set in `NetgenParams` take a list of numbers as an argument.

- `RefinementPoints` : Points in the 3D domain are specified together with the desired refinements at those points. They are listed as follows,

  ```
  {RefinementPoints : [p1.x,p1.y,p1.z,d1,
  p2.x,p2.y,p2.z,d2] }
  ```

  This list sets a local mesh size of `d1` at point `p1` and `d2` at point `p2`. In this way, any number of points and the local mesh size can be set.

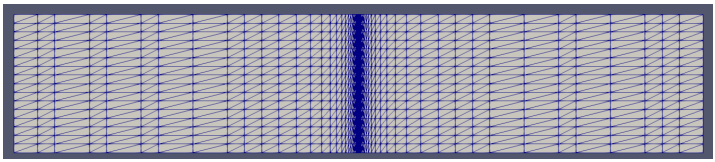
- `RefinementLines` : Lines in the 3D domain are specified together with the desired refinements along those lines as follows,

  ```
  {RefinementLines : [p1.x,p1.y,p1.z,p2.x,p2.y,p2.z,d1,
  p3.x,p3.y,p3.z,p4.x,p4.y,p4.z,d3]}
  ```

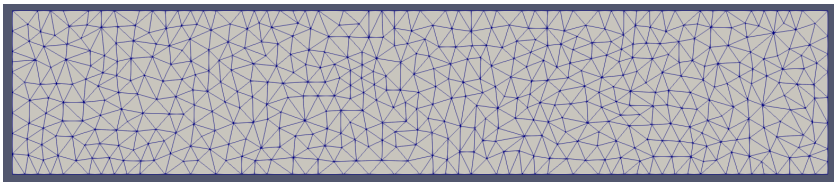
  This list sets a local mesh size of `d1` along the line `p1` and `p2` as well as a mesh size of `d3` along the line between `p3` and `p4`. In this way, any number of lines and corresponding local mesh size can be set.

6.4 Python interface - mesh generation

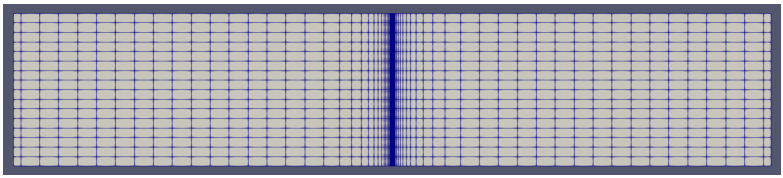
The `mesher` (or `tensormesher`) provides following command for mesh generation and retrieving mesh data.



(a) FEM

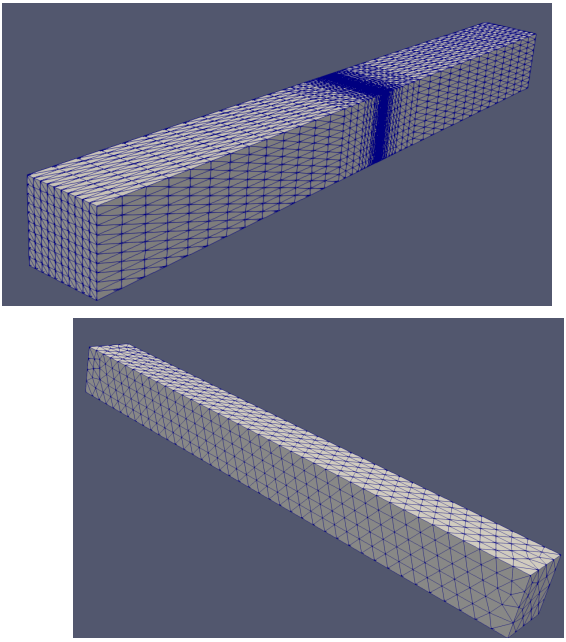


(b) TetMesh

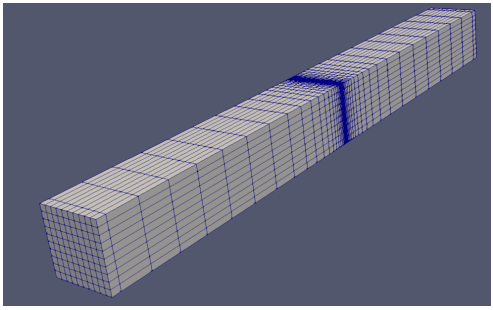


(c) Tensor

Figure 6.1: Mesh of a 2D diode generated by running the diode config file in Chapter 3 with MeshType of (a) FEM, (b) TriMesh, (c) Tensor.



(b) 3D-TetMesh



(c) 3D-Tensor

Figure 6.2: Mesh of a 3D diode generated by running the diode config file in Chapter 3 with MeshType of (a) FEM, (b) TetMesh, (c) Tensor.

6.4.1 Create mesh

The command `dio.createDeviceMesh()` creates mesh from scratch. For this command to work properly, no mesh must be present in the device. Hence, it is necessary to run `dio.clearDeviceMesh()` command to erase existing mesh.

6.4.2 NetGen mesh settings

The command `dio.setNetgenParams()` provides *NetGen* global meshing parameters. It takes two arguments as follows.

1. `NumericParams` : A Python dictionary mapping a parameter to its numeric value is specified. Following parameters can be passed in the dictionary.
    - `maxh` : Maximum global mesh size allowed.
    - `grading` : Mesh grading quality - 0: uniform, 1: local grading enabled.
    - `fineness` : Mesh refinement quality - 0: coarse, 1: refined.
    - `maxiter_refine` : Maximum number of iterations in 'iterative refinement'.
    - `optsteps_3d` : Number of optimization runs for 3d meshing.
2. `NumericListParams` : A Python dictionary mapping a parameter to a python list of numbers is provided. Following parameters can be passed in the dictionary.
    - `RefinementPoints` : Points in the 3D domain are specified together with the desired refinements at those points. They are listed as follows - `[p1.x,p1.y,p1.z,d1,p2.x,p2.y,p2.z,d2]`. This list sets a local mesh size of `d1` at point `p1` and `d2` at point `p2`. In this way, any number of points and the local mesh size can be set.
    - `RefinementLines` : Lines in the 3D domain are specified together with the desired refinements along those lines as follows, `[p1.x,p1.y,p1.z,p2.x,p2.y,p2.z,d1,p3.x,p3.y,p3.z,p4.x`

This list sets a local mesh size of `d1` along the line `p1` and `p2` as well as a mesh size of `d3` along the line between `p3` and `p4`. In this way, any number of lines and corresponding local mesh size can be set.

6.4.3 Clear mesh

The command `dio.clearDeviceMesh()` clears existing device mesh, if present.

6.4.4 Clear structure and mesh

The command `dio.clearDeviceData()` clears existing device structure (i.e. all the regions, doping definitions, and contacts) and mesh.

6.5 Python interface - mesh data retrieval

Mesh data may be needed to postprocess results, such as integration or averaging of a vertex physical quantity (e.g. eletron density). Following commands are provided in the python interface for retrieving `tensormesher` data. These commands are called on the `device` python object. Note, that a tensor mesh is a rectangular/cubic mesh completely specified by the grid spacing along X-, Y-, and Z- (in 3D) axes.

6.5.1 Get region information

The command `getVertexRegionMapNp()` returns region ids for each point on the grid. It is returned as a "numpy array" of integers. Length of the array is the number of vertices in the device. Each integer represents a region id.

Region names and material corresponding to all the region ids can be retrieved by `getRegionNames()` and `getRegionMaterials()` commands, respectively. They are retured as a python list of strings (words) in the same order as region ids.

6.5.2 Tensor mesh-grid spacing

Grid spacings along X-, Y- and Z- axes are returned by the following commands-

- `getTensorMeshXLinSpaceNp()`
- `getTensorMeshYLinSpaceNp()`
- `getTensorMeshZLinSpaceNp()` in 3D. In 2D devices, `getTensorMeshZLinSpa` returns Y-spacing.

Note, that the grid spacings are returned as a "numpy array".

6.5.3 Tensor mesh vertex id

The command `getNumberOfVertices()` returns the number of vertices in the structure.

Vertex id of the vertex at x-, y-, z- point indices `ix`, `iy`, `iz` is returned by `getVidFromXidYidZid(ix,iy,iz)`.

Similarly, `ix`, `iy`, `iz` point indices corresponding to the given vertex id `vid` are returned by `getXidYidZidFromVid(vid)`. They are returned as python list of three integers.

The x-, y-, z- coordinates of the vertex `vid` are given by $(x[ix], y[iy], z[iz])$, where x, y, z are X-, Y-, and Z- line-spacings returned by the commands mentioned before, and ix, iy, iz are point indices.

6.5.4 FEM mesh vertex coordinates

The command `getNumberOfVertices()` returns the number of vertices in the structure.

The x-, y-, z- coordinates of the vertex of the finite element mesh are given by the following commands,

- `getXsNp()`
- `getYsNp()`
- `getZsNp()` in 3D. In 2D devices,`getZsNp()` returns Y-spacing.

Note, that the vertex coordinates are returned as a "numpy array" of length equal to the number of vertices. Thus, coordinates of `vid`th vertex are given by $(x[vid], y[vid], z[vid])$, where x, y, z are X-, Y-, and Z- coordinate lists returned by the commands mentioned above.

6.5.5 FEM mesh element list

The command `getNumberOfElements()` returns the number of triangular (in 2D mesh) or tetrahedra (in 3D mesh) elements in the structure.

Each triangle or tetrahedral element consists of three or four vertices, respectively. The command `getElementListNp()` returns a 2D "numpy" array of integers of dimension - `numElements` $\times 3$ (2D mesh) or `numElements` $\times 4$ (3D mesh). The leading dimension lists the elements while the trailing dimension lists the vertex ids corresponding to the element. For example, all vertices (3 or 4) of i^{th} element are given by the array $e[i]$, while j^{th} vertex of i^{th} element is given by $e[i][j]$.

6.5.6 Voronoi volume

Volume in μm^3 (area in μm^2 in 2D) of voronoi region around the given vertex id `vid` is returned by `getVoronoiVolumeAtVertex(vid)`.

Only in FEM mesh, the command `getVoronoiVolume()` returns a numpy array of voronoi volume at all the vertices in the device.

Chapter 7

Structure Generation from GDS Layout files

Fabrication of semiconductor devices on the chip is performed by repeated application of implantation, etching, oxidation, or annealing steps. Masks are often used to selectively perform implantation or etching in semiconductor regions to create pn-junctions or to isolate devices. Mask files are often stored in GDS format. To create these devices for simulation, GDS mask files need to be imported into the structure generator. This chapter describes how to import GDS mask files to transfer masks into the device structures.

7.1 Config File

A GDS mask file named 'coupler.gds' stores mask of a coupled waveguide structure as shown in Fig.7.1. The GDS file consists of a set of convex 2D polygonal objects known as 'layers'. These layers are used to manufacture a photo-lithography mask. Among all the layers, the group of layers which are used to create a certain semiconductor device are often bundled together in a layer group called 'cell'. The cell can be used to repeat the mask designs at multiple locations on the chip to create devices.

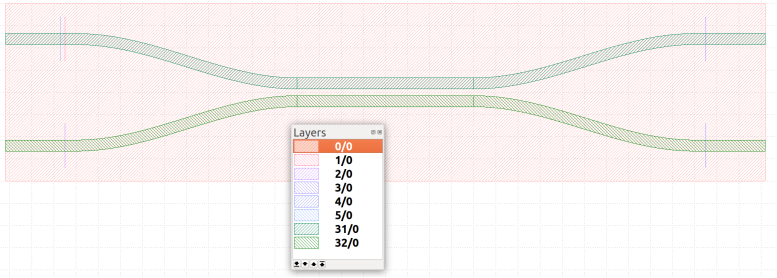


Figure 7.1: Example GDS file drawing the design of two Silicon waveguides on Oxide layer. The waveguides are coupled in the middle section by bringing them closer. This file is taken from []. Layer numbers are provided in the inset. The file consists of only a single cell named 'test'.

Following config file imports the GDS file and creates 3D structure of a coupled waveguide.

```
Device:
{
Name = "SiWG";
MeshType = "FEM";
Simulation = "DD";
}

GDSLayerDef*C1:
{
GDSFileName = "coupler.gds";
Layer*L1: {Cell="test", Layer=0, Y=0, dY=0.2};
Layer*L2: {Cell="test", Layer=31, Y=0.2, dY=0.2};
Layer*L3: {Cell="test", Layer=32, Y=0.2, dY=0.2};
Layer*L4: {Cell="test", Layer=0, Y=0.2, dY=0.2};
}

RefWin*RefAll:
{
Position: ([-18., -.1, -6.], [18., 0.5, 6.]);
```

```
Shape = "Cuboid";
}

Region*RegGas:
{
RefWin = ["RefAll"];
Material = "Gas";
}

Region*RegSi:
{
        // activate for a region of the coupler shape
//RefWinGDS = ["C1_L2", "C1_L3"];
        // activate for the doping of the coupler shape
RefWinGDS = ["C1_L4"];
Material = "Silicon";
}

Region*RegOxTop:
{
RefWinGDS = ["C1_L1"];
Material = "Oxide";
}

/*DopingDef*Pdoping:
{
RefWin = "RefAll";
Type =  "Gaussian";
Concentration = 5E16;
Depth = 0.1;
DecayLength = 0.1;
BaseWinGDS = ["C1_L2", "C1_L3"];
Dopant = "Boron";
}*/

MeshDef*MDefAll:
{
  RefWin = "RefAll";
```

```
  Xarg: [ 0., 0.0001, 0.05, 1.3 ]
  Yarg: [0., 0.025, 0.025, 1. ]
  Zarg: [0., 0.025, 0.025, 1. ]
}
```

Among the objects listed in the above config file, `Device` (see Sec. 2.1.1), `RefWin` (see Sec. 2.1.2), `MeshDef` (see Sec. 2.1.6), and `Region` (see Sec. 2.1.3) have the same functionality as explained in Chapter 3. Differences in some of the objects are described below.

7.2 GDSLayerDef object

A `GDSLayerDef` object reads the gds file and creates 2D or 3D RefWins from the mask layers, which can be used to create various regions. Name of the object is specified as a string after 'GDSLayerDef*' string. In this example, the `GDSLayerDef` is named 'C1'. The GDS file is specified by `GDSFileName`. A `Layer` object in `GDSLayerDef` reads a specific layer and converts it to a 2D or 3D RefWin depending on the inputs as follows.

7.2.1 Layer object

A `Layer` object sets which layer from the GDS file is read using the argument `Layer` from the cell specified using the argument `Cell`. Each layer is composed of a group of 2D polygonal mask objects which are assumed to be in XZ plane in mesher object. The argument `Y` sets y-intercept of the XZ plane of these 2D masks in the device. If the argument `dY` is not set or is set to 0, then 2D 'RefWin' of the 2D mask polygons is created. If `dY` is set to a negative number, the 2D polygons are *extruded* towards negative Y, and a 3D RefWin is created in the polygonal volume in XZ plane and $\mathrm{Y} - \mathrm{dY} \leq y \leq \mathrm{Y}$. If `dY` a positive number, the 2D polygons are *extruded* towards positive Y, and the 3D RefWin is created in the volume in XZ plane and $\mathrm{Y} \leq y \leq \mathrm{Y} + \mathrm{dY}$.

RefWin created by the above operations is noted as `RefWinGDS`. It is named '<GDSName>_<LayName>', where <GDSName> is the name of `GDSLayerDef` object and <LayName> is the name of `Layer` object. For example, Layer object named 'L1' in GDS layer definition

'C1' creates a RefWin 'C1_L1'. It can be referenced from `Region` object using this name.

Multiple `GDSLayerDef` objects with different names can be created in a config file which enables importing multiple GDS files to create a device structure.

7.3 Region object

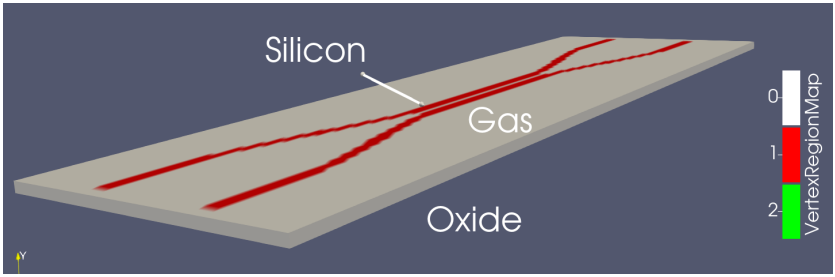
When a mask layer is used for masked etching or blanket deposition of a material layer followed by masked etching, a 3D region of the shape of the mask with a specific thickness is formed on the substrate. Such a region can be created in the structure by importing the relevant mask, specifying `Y` and `dY` to create a 3D RefWinGDS, and linking this RefWinGDS with the region object. A usual `Region` object is created and its material is set normally. Then, names of all the linked RefWinGDS are specified as a comma separated list with the argument `RefWinGDS`. Note, `RefWin` argument must not be set when `RefWinGDS` is provided.

A region of the shape of the coupler is created by un-commenting the line `RefWinGDS=["C1_L2","C1_L3"]` and commenting out the line `RefWinGDS=["C1_L4"]` in `RegSi` region and running the above config file to generate the mesh. The generated mesh can be viewed in `paraview` using 'SiWG_str.xdmf' file. The device is shown in Fig. 7.2.

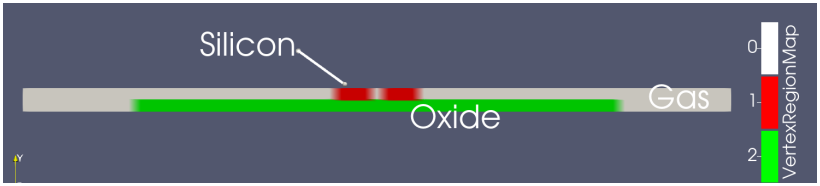
A similar structure can be created with the tensor mesh by setting `MeshType="Tensor"`. Tensor mesh can be created by running the mesher and can be visualized in `paraview`. The device structure is shown in Fig. 7.2(a) and cutline is shown in Fig. 7.2(b).

7.4 DopingDef object

A mask is typically used for implantation of various dopant ions into the semiconductor substrate. Implantation and subsequent annealing creates a vertical Gaussian profile of dopants in the unmasked areas of the substrate, together with lateral straggle under the masked areas. This effect can be mimicked by creating a `DopingDef` object as described in Sec. 5.3, and passing the 2D mask RefWins (created by



(a) Meshing



(b) Doping

Figure 7.2: A region object shown in the figure is created when a RefWinGDS corresponding to the region in the shape of the coupler is activated. In the above file, '0' corresponds to 'RegGas', '1' corresponds to the 'RegSi', and '2' corresponds to the 'RegOxTop'.

setting `dY=0.`) as comma separated list to the argument `BaseWinGDS` in `DopingDef` object.

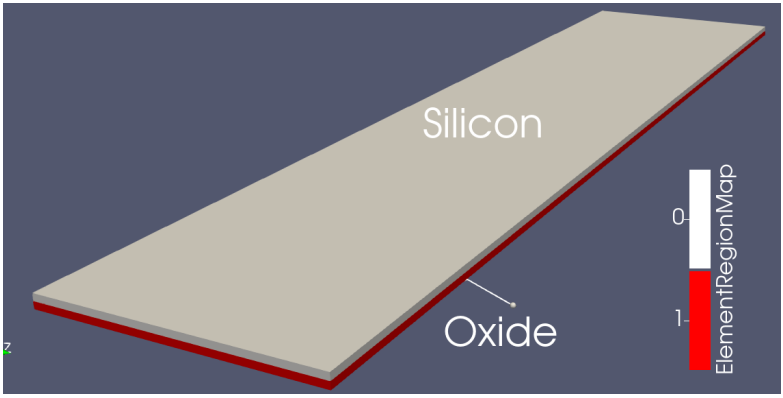
A doping profile of the shape of the coupler is created by commenting out the line `RefWinGDS=["C1_L2","C1_L3"]` and un-commenting out the line `RefWinGDS=["C1_L4"]` in `RegSi` region. Also un-comment the doping profile object. Modify the `GDSLayerDef` object as follows.

```
GDSLayerDef*C1:
{
GDSFileName = "coupler.gds";
Layer*L1: {Cell="test", Layer=0, Y=0, dY=0.2};
Layer*L2: {Cell="test", Layer=31, Y=0.2, dY=0.0};
Layer*L3: {Cell="test", Layer=32, Y=0.2, dY=0.0};
Layer*L4: {Cell="test", Layer=0, Y=0.2, dY=0.2};
}
```

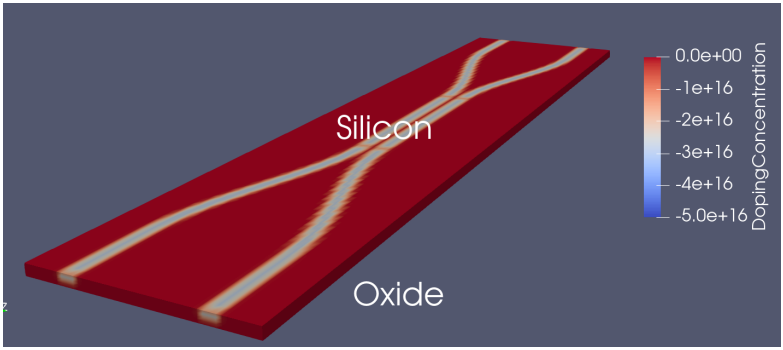
Since the layers `L2` and `L3` are used as a `BaseWin` (or in other words, a mask), their thickness `dY` must be set to zero.

Now, run the above config file to generate the mesh. The generated mesh can be viewed in `paraview` using 'SiWG_str.xdmf' file. The device is shown in Fig. 7.3(a), and the doping profile is shown in Fig. 7.3(b). Cutplane perpendicular to the waveguide direction at the middle of the structure is shown in Fig. 7.3(c).

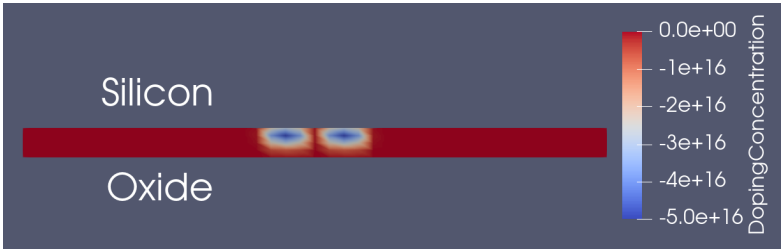
Note: A Python script interface is currently not provided for importing GDS files.



(a) Meshing



(b) Doping



(c) Contacts

Figure 7.3: (a) Device structure, (b) analytic doping profile as seen from the surface of the structure, and (c) doping distribution at the cutplane perpendicular to the waveguide direction at the middle of the structure.

Chapter 8

Graphical User-Interface

Device structure generation and meshing can also be performed with a graphical-user-interface (GUI) of the structure generator and mesher. The GUI can be opened using the following line -

```
meshergui &
```

The above command opens a GUI window which consists of a 'Menu-bar', a 'Tool-bar', a structure drawing board, and a side-pane. The 'Menu-bar' contains various functions as drop-down menus, while the 'Tool-bar' contains some of the more common functions as short-cut buttons. The 'side-pane' contains a stack of different widgets. Each of the widgets contains an information on the structure, GDS layout to structure conversion, python script for generation, etc. The shapes added to the structure appear on the structure drawing board. The `meshergui` tool saves the current structure into a file with extension '*.sstr'. This file can be opened later in the `meshergui` tool to load the structure.

The **meshergui** performs finite-element meshing of the device.

The **tensormeshergui** performs cubic or rectangular meshing of the device.Apart from that, both the programs have the same GUI.

Additionally, the **processgen** provides GUI for semiconductor process modeling and simulation. Some of the menus of this program are also present in *disabled state* in the above mentioned mesher programs.

Various functionalities available as a single-click buttons on the GUI window are described in Fig. 8.1. Each of the above functions has been described in detail below.

8.1 File-menu

'File' menu on the menu-bar shows the following 'items' on click (see Fig. 8.2).

8.1.1 New

When clicked, it opens a new `meshergui` window keeping the current circuit unmodified. *If the current window is empty, then it does nothing.*

8.1.2 Open

When clicked, it opens a dialog box to select the '*.sstr' file to be opened. The selected '*.sstr' file is opened in a new `meshergui` window. *If the current window is empty, then the circuit file is loaded to the current window instead.*

8.1.3 Import GDS File

When clicked, it opens a dialog box to input file-name and location of the '*.gds' file to be imported. The layout file in GDS format is imported and is then available in the tab named `GDS Layout to Structure`.

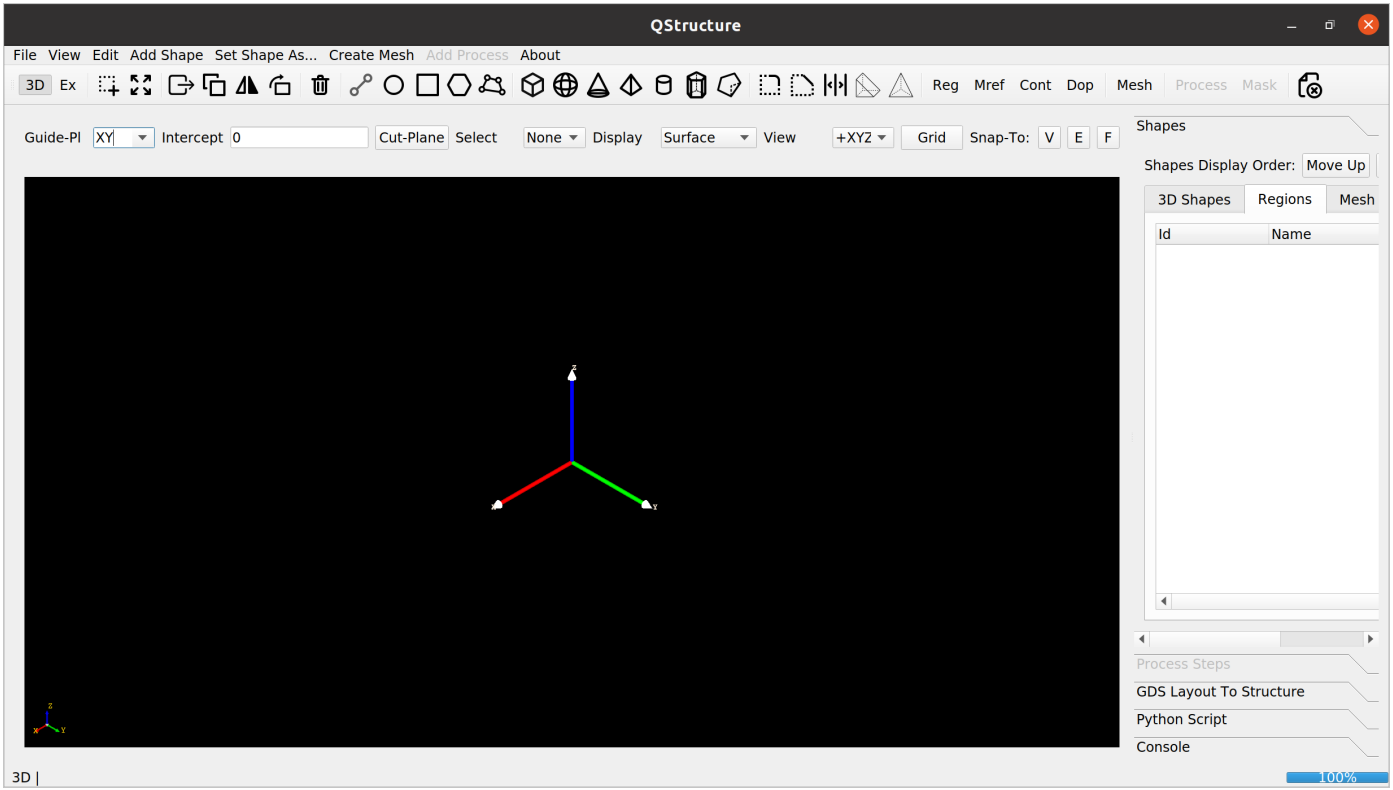


Figure 8.1: Graphical-user-interface of the structure generation tool.

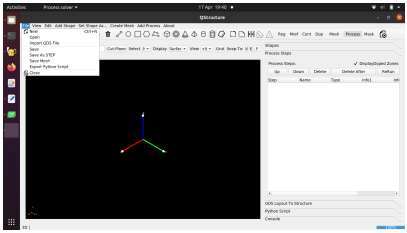


Figure 8.2: File menu.

8.1.4 Save

When clicked, it opens a dialog box to input file-name and location of the '*.sstr' file to be saved. Following information is stored in the '*.sstr' file with the given name and at the given location.

- Defined RefWins
- Defined regions, mesh definitions, contact definitions, and doping definitions
- Python script to generate the structure

It does not store generated mesh, nor does it store imported GDS file. User is requested to recreate mesh after opening the *.sstr file.

Working directory: The directory from which processgen is called, is the default working directory. However, the folder in which the structure is stored automatically becomes a working directory.

Device name: Name of the file (excluding extension) is the device name. Other files (e.g. mesh file, python file) begin with the same name in the working directory. There is *no* default name for the device.

8.1.5 Save As

When clicked, it opens a dialog box to input file-name and location of the '*.sstr' file to be saved. Current state of the `meshergui` is stored.

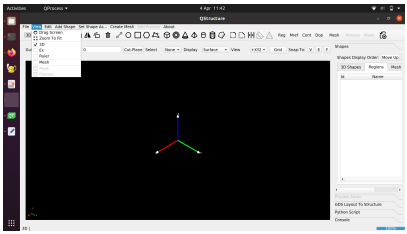


Figure 8.3: View menu.

8.1.6 Save As STEP

When clicked, it opens a dialog box to input file-name and location of the '*.step' file to be saved. All the shapes linked to the regions are stored in the step file.

8.1.7 Export Python Script

When clicked, it opens a dialog box to input file-name and location of the python script to be saved. Python script generated in the tab named `Python Script` is stored in the file. If the structure is previously stored, the python script is stored in the working directory with the same name as the device name.

8.1.8 Save Mesh

When clicked, it stores the mesh file 'devicename.h5' and the xml script 'devicename.xdmf' in the working directory. Here, `devicename` is the name of the device stored before beginning meshing. The 'devicename.xdmf' mesh file can be viewed in `paraview`.

8.2 View-menu

'View' menu on the menu-bar shows the following 'items' on click (see Fig. 8.3).

8.2.1 Zoom-to-fit

When clicked, adjusts zoom such that the entire structure is within the window.

8.2.2 3D

When clicked, it toggles '3D'-button. Deactivate '3D'-button to draw 2D structures, and activate it to draw 3D structures. Note, that the button can be toggled only when no shape is present. Thus, it is advisable to toggle it to suitable state before drawing the structure.

8.2.3 Ex

When clicked, it toggles 'Ex'-button. When 'Ex' is active, exact coordinates of the corners of the circumscribing cuboid are requested from the user while drawing the shapes.

8.2.4 Ruler

When clicked, 'Ruler'-button toggles. When 'Ruler' is active, distance can be measured between any two points on the structure. To do so, click on the first point. A ruler starting at that point will be dynamically displayed. Clicking the second point will fix the ruler and display distance between the two point.

8.2.5 Mesh

When clicked, 'Mesh' button toggles the view between mesh-view and structure-view. If mesh is not generated, mesh-view will display an empty window. *Note, that when mesh-view is active, the structure cannot be edited.*

8.3 Edit-menu

'Edit' menu on the menu-bar shows the following 'items' on click (see Fig. 8.4).

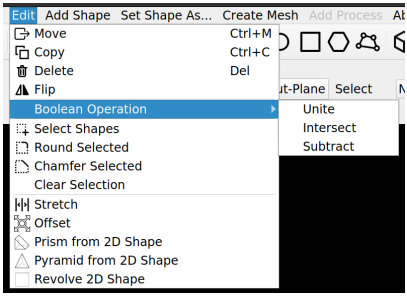


Figure 8.4: Edit menu.

8.3.1 Move or Copy

Click on 'Move' or 'Copy' buttons to toggle it. When 'Move'/'Copy' is active, click on any shape to begin moving it. Moving the cursor will move the shape together with the cursor. Click on the destination point to place the shape.

If 'Copy' is active, a copy of the initial object will be placed at the destination without changing the initial shape.

8.3.2 Delete

Click on 'Delete' button to toggle it. When 'Delete' is active, click on any object to delete it. Note, the shape will be deleted only if it is *not* assigned to any of the `Region`, `MeshDef`, `DopingDef`, or `Contact`. Else, first delete the respective entities first before deleting the shape.

8.3.3 Flip

Click on 'Flip' button to toggle it. When active, click on any shape to *mirror* it. The shape is reflected relative to the current guide-plane (`XY`, `YZ`, or `XZ`) placed at current intercept along the plane-normal.

8.3.4 Select shapes

Click on 'Select shapes' button to toggle it. When active, click on any geometric shape to select it. Depending on the the selected option

in 'Select' drop-down menu in 'display-control' panel, the different geometric shapes are selected as follows.

- `None` : Any of the solid entity is selected.
- `Body` : A 3D shape is selected.
- `Face` : A 2D face of the 3D structure *or* a 2D shape is selected.
- `Edge` : An edges of a 2D or 3D shape is selected.
- `Point` : A corner point of a 2D or 3D shape is selected.

Pressing Escape key or clicking 'Clear selection' button empties the selected shapes list. Alter If you wish to perform subsequent actions on the selected shapes, please don't press Escape key or click 'Clear selection'.

8.3.5 Boolean operations - Unite, Intersect, and Subtract

Select any set of shapes. Then click on any of the three boolean operations. The following actions will happen.

- `Unite` : The selected shapes will be united with the *first* selected shape.
- `Intersect` : Common area between *first* selected shape and the next selected shapes is computed and set to the first shape.
- `Subtract` : The selected shapes will be subtracted from the *first* selected shape.

Notice, that *the first selected shape is modified at the end of each of the above boolean operations.*

8.3.6 Round Selected or Chamfer Selected

To round *all* the corners of the shapes with a specific rounding radius, first select them. Then click on 'Round selected'. A dialog box will

open up asking for rounding or Chamfer radius. Specify the radius and press Ok.

To round *specific* corners or edges of the shapes, first select these corners/edges by using `Point / Edge` option in 'Select' drop-down menu in 'display-control' panel. Then click on 'Round selected'. A dialog box will open up asking for rounding or Chamfer radius. Specify the radius and press Ok.

8.3.7 Stretch

To stretch the given shape along a specific major axis, activate 'Stretch' button. When active, click on any of the shapes. A dialog box will appear. Specify stretch length and press ok. The given shape will be stretched at the clicked point along the axis normal to the current guide-plane.

8.3.8 Offset

To offset the given shape by a specific offset distance, activate 'Stretch' button. When active, click on any of the shapes. A dialog box will appear. Specify offset distance and press ok. The given shape will be offset by the given distance. This could be useful in mimicking deposition process.

8.3.9 Prism from 2D Shape

In 3D mode, activate 'Prism from 2D Shape' button. When active, click on the 2D shape which forms base of the prism. Then click on corresponding point which is at the top surface of the prism. A prism will be drawn by 'sweeping' the base along the vector starting at the base point and ending at the top surface point.

8.3.10 Pyramid from 2D Shape

In 3D mode, activate 'Pyramid from 2D Shape' button. When active, click on the 2D shape which forms base of the prism. Then click on the *apex* point of the pyramid. A pyramid will be drawn by connecting all the corners of the base to the apex point.

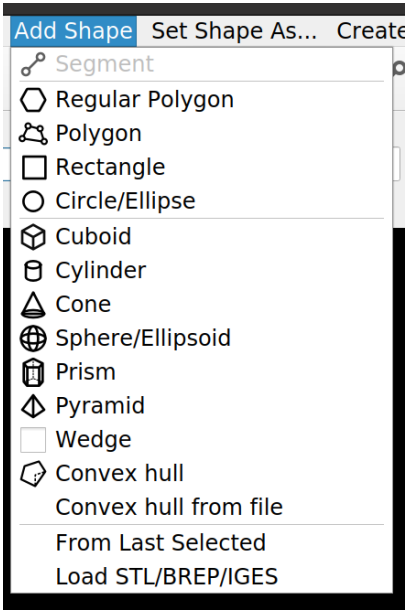


Figure 8.5: Add shape menu.

8.3.11 Revolve 2D shape

In 3D mode, activate 'Revolve 2D shape' button. When active, click on the 2D shape which is to be revolved. The axis normal to the current guide-plane (`XY`, `YZ`, or `XZ`) is the axis of revolution. Click on any of the point to fix the axis of revolution at any point the plane. The 2D shape will be revolved around the axis at the given point.

8.4 Add-Shape-menu

'Add Shape' menu on the menu-bar enables selecting various shapes to draw in the window. It lists the following 'items' on click (see Fig. 8.5).

8.4.1 Segment

When active, successively click on the start-point and end-point of the segment to add a segment to the shapes. It is disabled in 3D mode.

Following buttons add 2D-shapes to the drawing board.

8.4.2 Axis-aligned 'Rectangle'

When active, successively click on a corner and a diagonally opposite corner of the 'axis-aligned' rectangle to add a rectangle to current guide-plane in the drawing board.

8.4.3 Circle

When active, successively click on a corner and a diagonally opposite corner of the 'axis-aligned' rectangle. This will add a circle or an ellipse 'in-scribing' the axis-aligned rectangle to current guide-plane in the drawing board.

8.4.4 Regular Polygon

When active, successively click on a corner and a diagonally opposite corner of the 'axis-aligned' rectangle. This will add a regular polygon, which in-scribes the axis-aligned rectangle to current guide-plane in the drawing board.

8.4.5 Polygon

When active, successively click on a consecutive corners of the polygon in clockwise or counter-clockwise direction. A polygon will be drawn connecting the added points. A double-click on the point will close the polygon. This operation will add a polygon to current guide-plane in the drawing board.

Following buttons add 3D-shapes to the drawing board. They are disabled when 3D is disabled.

8.4.6 Cuboid

A 'axis-aligned' cuboid is drawn in two steps. First, a 'axis-aligned' rectangle at the cuboid base is drawn by successively click on a corner and a diagonally opposite corner of the rectangle in the drawing board. In the second step, click on the top corner of the cuboid will add a 'axis-aligned' cube to the drawing board.

8.4.7 Cylinder

A 'axis-aligned' cylinder is drawn in two steps. First, a 'axis-aligned' circle at the base is drawn by successively click on a corner and a diagonally opposite corner of the circumscribing rectangle in the drawing board. In the second step, click on a point on the top of the cylinder will add a cylinder to the drawing board, whose base is on the guide-plane.

8.4.8 Cone

A cone is drawn in two steps. First, a 'axis-aligned' circle at the base is drawn by successively click on a corner and a diagonally opposite corner of the circumscribing rectangle in the drawing board. In the second step, click on a point on the top face of the cone will add a cone to the drawing board, whose base is on the guide-plane.

8.4.9 Sphere/Ellipsoid

A ellipsoid is drawn in two steps. First, a 'axis-aligned' circle at the cross-section of the ellipsoid is drawn by successively click on a corner and a diagonally opposite corner of the circumscribing rectangle in the drawing board. In the second step, click on a point on the top face will add a ellipsoid to the drawing board.

8.4.10 Prism

When clicked, a dialog box appears asking number of sides of the prism. The number of sides remain fixed for drawing subsequent prisms. To change them, deactivate and reactivate 'Prism' button.

The prism is drawn in two steps. First, a 'regular' polygon at the base of the prism is drawn by successively click on a corner and a diagonally opposite corner of the circumscribing rectangle in the drawing board. In the second step, click on a point on the top face of the prism will add a prism to the drawing board, whose base is on the guide-plane.

8.4.11 Pyramid

When clicked, a dialog box appears asking number of sides of the base of the pyramid. The number of sides remain fixed for drawing subsequent prisms. To change them, deactivate and reactivate 'Pyramid' button.

The pyramid is drawn in two steps. First, a 'regular' polygon at the base of the pyramid is drawn by successively click on a corner and a diagonally opposite corner of the circumscribing rectangle in the drawing board. In the second step, click on a point at the apex will add the pyramid to the drawing board, whose base is on the guide-plane.

8.4.12 Convex Hull

When active, successively click on a consecutive corners of the polyhedron. Double-click one all the points are clicked. A convex hull of all the added points will be constructed and added to the drawing board.

8.4.13 Convex Hull From File

When clicked, a dialog box requesting file name of the *.csv file open up. The *.csv file contains x, y, z coordinates of the points as comma separated list, one point per line. Thus, the csv file is a $N \times 3$ table. A convex hull of these points is constructed and added to the drawing board.

8.4.14 From Last Selected

Select any shape or its face or edge. After that, click 'From Last Selected'. The selected shape is added to the structure as a new shape.

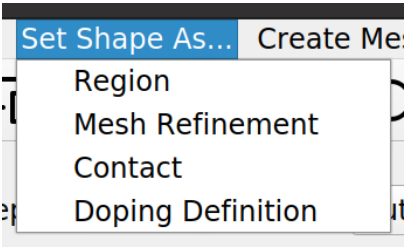


Figure 8.6: 'Set Shape As...' menu.

8.4.15 Load STL/BREP/IGES Files

Shapes stored in STL, BREP or IGES files are loaded and added to the structure.

8.5 Set Shape As...

'Set Shape As...' menu on the menu-bar enables creation of `Regions`, `MeshDefs`, `Contacts`, and `DopingDefs`. It lists the following 'items' on click (see Fig. 8.6).

Before creating any of the above entities, one or more shapes must be selected by using 'Select Shapes' button.

8.5.1 Region

Clicking 'Region' opens a dialog box which requests additional parameters including material to define the region. Notice, that the names of selected `RefWins` are listed as drop-down list in the dialog box. You can highlight each RefWin (for your information) by selecting it in the drop-down list. Once the additional parameters are set, click Ok to add the region to the structure.

8.5.2 Mesh Refinement

Clicking 'Mesh Refinement' opens a dialog box which requests additional parameters to define the quadtree/octtree mesh refinement.

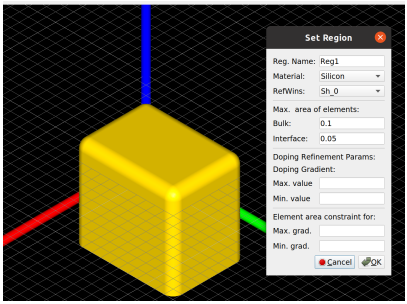


Figure 8.7: The dialog box opening up after clicking 'Region' is shown above.

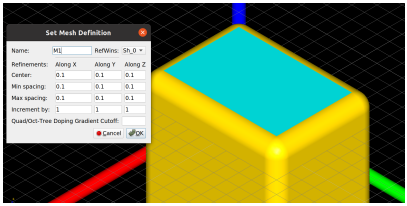


Figure 8.8: The dialog box opening up after clicking 'Mesh Refinement' is shown below.

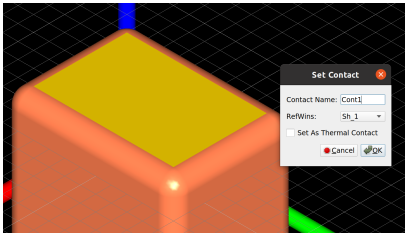


Figure 8.9: The dialog box opening up after clicking 'Contact' is shown below.

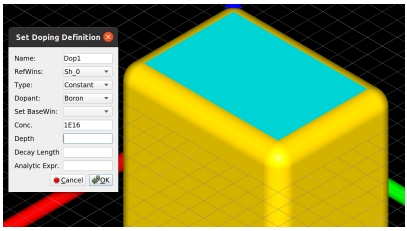


Figure 8.10: The dialog box opening up after clicking 'Doping Definition' is shown below.

Notice, that the names of selected `RefWins` are listed as drop-down list in the dialog box. Once the additional parameters are set, click Ok to add the mesh refinement to the structure.

8.5.3 Contact

Clicking 'Contact' opens a dialog box which requests additional parameters to define the contact. Notice, that the names of selected `RefWins` are listed as drop-down list in the dialog box. You can highlight each RefWin (for your information) by selecting it in the drop-down list. Once the additional parameters are set, click Ok to add the mesh refinement to the structure.

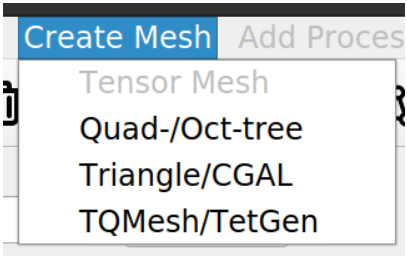


Figure 8.11: 'Create Mesh' menu.

8.5.4 Doping Definition

Clicking 'Doping Definition' opens a dialog box which requests additional parameters to define the doping definition. Once the additional parameters are set, click Ok to add the doping definition to the structure.

8.6 Create Mesh

'Create Mesh' menu on the menu-bar enables creation of `QuadTree`/`OctTree` or `Triangle`/`TetGen` or `Tensor` mesh. It lists the following 'items' on click (see Fig. 8.13).

- `Tensor`: Create a tensor-mesh. It is enabled when `meshergui` for tensor structures is opened.
- `QuadTree` or `OctTree` : Create a FEM-mesh using quadtree (2D structures) or octtree (3D structures) based meshers.
- `Triangle` or `TetGen` : Create a FEM-mesh using Triangle mesher (2D structures) or TetGen mesher (3D structure).

On successful meshing, generated mesh is displayed on the drawing board. You can toggle the view to hide mesh and display the device structure by clicking on Mesh.

In the next part, functionalities of the tabs on the side-pane are described.

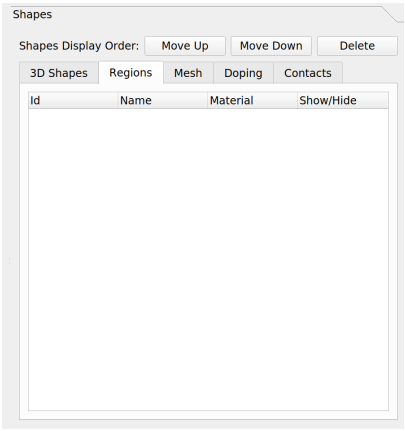


Figure 8.12: 'Shapes' tab in the side-pane.

8.7 Shapes-Tab

All the `RefWins` (called shapes in this context), `Regions`, `MeshDefs`, `DopingDefs`, and `Contacts` created by the structure generator are listed in each of the tabs here row-wise. Notice, that fourth column in each of these tabs has a header named 'Show/Hide'. Check-boxes corresponding to each of the entities are listed in this column. Unchecking (checking) the checkbox hides (shows) the shape corresponding to this entity.

To show or hide *all* the `Regions`, click the header. It toggles display state of all the shapes. Similarly, *all* the `MeshDefs`, `DopingDefs`, and `Contacts`, can be toggled by clicking on the header.

Order of the regions listed in the tab is important. Regions listed below in the list are subtracted from those listed above. Same rule is applied for the `MeshDefs`, `DopingDefs`, and `Contacts`. Selected entity in the list can be moved up or down by Move Up and Move Down buttons, respectively. Selected entity can be deleted by using Delete button.

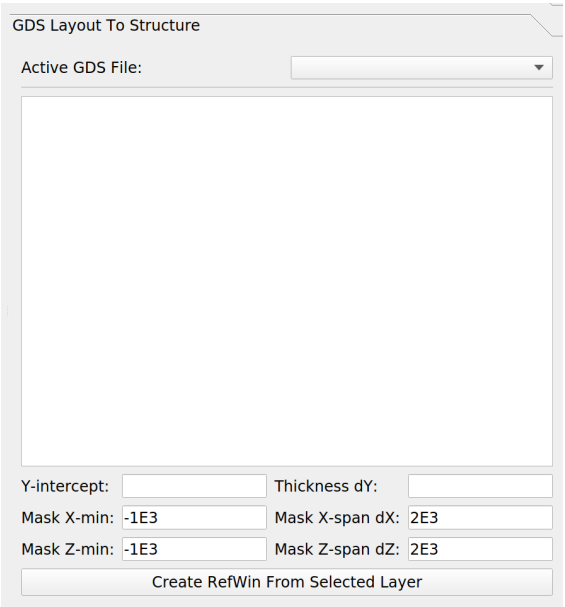


Figure 8.13: 'GDS Layout To Structure' tab in the side-pane.

8.8 GDS Layout To Structure

Chip layouts in GDS format can be used to create shapes. A GDS layout file is imported by clicking File → Import GDS File. All the GDS files imported in this session are listed in the drop-down box of `Active GDS File`. All the layers defined in the selected GDS file are listed in this tab. Checkbox in each row selects whether the given structure is visible or not.

A new shape is created from the selected layer specifying the parameters `Y intercept`, `Thickness`, mask `Xmin`, `dX`, `Zmin`, `dZ`. After adding correct input parameters, click on the Create RefWin button. A new shape will be created. Please refer to Chapter 7 to know more about the parameters and their use.

Appendix A

Notation and Acronyms

Acronyms

BPM	Beam Propagation Method
FDTD	Finite Difference Time Domain
NEGF	Non-equilibrium Green's Function
QT	Quantum Transport

Bibliography