

A photograph of a power plant at sunset. The sky is a deep, vibrant orange, and thick plumes of white smoke or steam rise from several chimneys, silhouetted against the bright light. The foreground shows the dark silhouettes of trees and the power plant's structures.

Understanding OghmaNano

Roderick C. I. MacKenzie

Please do not cite this manual. Please see the section 1.6 on how to cite the model in your work.

Front cover: A picture of a thermal power station in Ratcliffe-on-Soar Nottinghamshire taken on a cold January afternoon in 2017. Most of the emissions you see in the image is water from the cooling towers however the gasses rising from the tall thin chimney on the left hand side of the image are the products of burning hydrocarbons which previously buried in the ground for about 300 million years.

This document is Copyright Roderick C. I. MacKenzie and compiled on May 15, 2023. If you have questions or comments please contact roderick.mackenzie@oghma-nano.com.

Contents

1 Introduction

- 1.1 What is OghmaNano?
- 1.2 Why OghmaNano?
- 1.3 About this book/manual
- 1.4 What is the history of OghmaNano?
- 1.5 What is the roadmap for OghmaNano?
- 1.6 Using OghmaNano in industrial/academic work
- 1.7 Bugs

2 Installing OghmaNano

- 2.1 Windows (if you have admin rights)
- 2.2 Windows (No admin rights)

3 Getting started

- 3.1 Simulating a JV curve of a simple solar cell
 - 3.1.1 Making your first simulation
 - 3.1.2 The output from your first simulation
 - 3.1.3 Editing device layers
 - 3.1.4 How do solar cells absorb light?
 - 3.1.5 Light inside solar cells
 - 3.1.6 Parasitic elements
 - 3.1.7 Solar cells in the dark
 - 3.1.8 The contact editor
 - 3.1.9 Electrical parameters

4 Simulation modes and simulation editors

- 4.1 JV editor (Steady state simulation editor)
 - 4.1.1 Inputs
 - 4.1.2 Outputs
 - 4.1.3 sim_info.dat
 - 4.1.4 Steady state electrical simulation
- 4.2 Time domain editor
- 4.3 Frequency domain editor
 - 4.3.1 Overview
 - 4.3.2 Inputs
 - 4.3.3 Outputs
- 4.4 Suns-Voc editor
 - 4.4.1 Outputs
- 4.5 Suns-Jsc editor
 - 4.5.1 Outputs
- 4.6 Quantum efficiency editor

4.6.1	Outputs	
4.7	Scanning probe microscopy editor	
4.8	Electrical equilibrium editor	
4.9	Steady state photoluminencense editor	
4.10	Charge extraction editor	
4.10.1	Outputs	
4.11	Capacitance voltage editor	
4.11.1	Outputs	
5	2D Simulations - OFETs	
5.0.1	The anatomy of a 2D simulation	
5.0.2	Electrical parameters	
5.0.3	Running a 2D simulation	
5.0.4	Meshing in 2D	
5.0.5	Solving the drift diffusion equations over the entire device	
6	2D simulation of bulk-heterojunctions	
7	Meshing	
7.1	Meshing	
7.2	Editing the electrical mesh/layers	
7.3	Should I be simulating in 1D, 2D or 3D?	
8	Theory of drift diffusion modelling	
8.1	Outline	
8.2	Electrostatic potential	
8.3	Free charge carrier statistics	
8.4	Carrier trapping and Shockley-Read-Hall recombination	
8.4.1	Equilibrium Shockley-Read-Hall recombination	
8.4.2	Non-equilibrium carrier trapping and recombination using Shockley-Read-Hall trap states	
8.4.3	Free-to-free carrier recombination	
8.4.4	Auger recombination	
8.5	Charge carrier transport	
8.6	Perovskite mobile ion solver	
8.7	Semiconductor interfaces	
8.7.1	Tunnelling through heterojunctions	
8.7.2	Doping on the interface	
8.8	Configuring the electrical solver	
8.8.1	Solver stability	
8.8.2	Simulating disordered devices without traps	
8.9	Calculating the built in potential	
8.9.1	Average free carrier mobility	
8.10	
8.10.1	Lattice thermal model	
8.10.2	Energy balance - hydrodynamic transport model	
9	Optical models	
9.1	
9.1.1	Lattice thermal model	
9.1.2	Energy balance - hydrodynamic transport model	

CONTENTS

9.1.3	Ray tracing model
10	Simple circuit simulations	
10.0.1	JV, IS, CV and other simulation modes
10.0.2	Using the fitting/scan tools with circuit models
11	Large area device simulation	
11.1	Designing contacts for large area devices
11.2	Simulating large area solar cells
12	Modelling excitons/geminate recombination - organics only	
12.1	Why you should not model excitons
12.2	Modelling excitons
12.3	Modeling excitons in a device
12.4	Modeling excitons in a unit cell
13	The oghma file format	
13.1	the .oghma simulation file format
13.2	Qwerks of the OghmaNano json format
13.3	Encoding
13.4	Forwards/backwards compatability of the file format
14	Databases	
14.1	Materials database
14.2	Adding new materials - the hard way
14.3	Adding new materials - the easy way
14.4	Emission database
14.5	Shape database
14.5.1	The shape file format
14.6	Filters database
14.7	Backups of simulations
15	Fitting experimental data	
15.1	Key tips and tricks
15.2	The main fitting window
15.3	Fit variables
15.4	How the fitting process works
15.5	Fitting without the GUI
16	Automation and Scripting	
16.1	The parameter scan window
16.1.1	Changing one material parameter
16.1.2	Duplicating parameters - changing the thickness of the active layer
16.1.3	Setting constants
16.1.4	The equivalent of loops
16.1.5	Limitations of the scan window
16.2	Multiparameter device optimizer
16.2.1	Using the multi parameter optimizer
16.3	Python/MATLAB scripting of OghmaNano
16.3.1	Python scripting
16.3.2	MATLAB scripting

17 Output files

- 17.1 Snapshots directory - dir
- 17.2 Trap_map directory - dir
- 17.3 Optical snapshots - dir
- 17.4 Cache - dir
- 17.5 Equilibrium directory
 - 17.5.1 Optical simulation
- 17.6 File formats
 - 17.6.1 .dat files
 - 17.6.2 .csv files
 - 17.6.3 Binary .csv files - files which are not human readable

18 Troubleshooting

- 18.1 Windows gives warns me the software is unsigned
- 18.2 Why don't I get a 3D view of the device

19 FAQ

- 19.1 Section
 - 19.1.1 Should I trust the results of OghmaNano?
 - 19.1.2 Can I use the model to simulate my exotic* material system/contacts?
- 19.2 Excited states

20 Legal

- 20.1 License
- 20.2 Copyright of the manual
- 20.3 Data privacy statement

Chapter 1

Introduction

1.1 What is OghmaNano?

OghmaNano officially stands for *Organic and hybrid Material Nano Simulation tool*. Oghma is also the name of the Gaelic God who's appearance is described as "sun-faced" or "shining/radiant", he is credited with developing Ogham, the script in which Irish Gaelic was first written. The creators of OghmaNano spent a lot of time making sure it can describe the light and optical radiation correctly thus the name seems like a good fit. How you remember the name is up to you.

OghmaNano was originally developed to be a general purpose model for simulating photovoltaic devices, including organic and perovskite cells. However, since its initial development the model has expanded to simulate many other classes of optoelectronic devices including, Organic Light Emitting Diodes (OLEDs), Organic Field Effect Transistors (OFETs), large area printed devices, optical filters, photonic crystals and many more. In general OghmaNano can simulate any opto-electronic-device where electrons, photons (and also heat - phonons) interact. The model has been downloaded by thousands of people across the globe (see figure 1.1) and is used in many top universities and companies. Figure 1.2 shows some of the classes of devices OghmaNano can simulate. Key features of OghmaNano are listed below:

- Electrical models:
 - 1/2D electrical drift-diffusion solver.
 - Dynamic SRH traps needed for simulating disordered materials.
 - Simple equivalent circuit model.
 - Complex 3D circuit model for complex large area devices.
 - Arbitrary user defined densities of trap states.
 - Thermal model linked to the electrical models.
 - Time domain, frequency domain and steady state solvers.

- Optical models:
 - Transfer matrix model for light
 - FDTD models
 - Ray tracing model
 - 1/2D optical mode solvers for waveguide structures.
 - Arbitrary light sources/filters.

- Excited states/mobile ion:
 - 1/2/3D Exciton solver.
 - Excited singlet/triplet state solver.
 - Mobile ions, doping and tunneling through interfaces.

- Other/databases:
 - Comprehensive materials databases.
 - Ability to convert arbitrary shapes to 3D objects.
 - Comprehensive 3D shape database.

1.2 Why OghmaNano?

By burning fossil fuels we are releasing ~ 33.3 gigatonnes of CO_2 per year [1] and thus humanity is steadily changing the composition of Earth's atmosphere. Since 1960, CO_2 in the atmosphere has risen by around 30% this in turn is increasing average global temperatures[2] and making our home planet Earth, a more difficult place to live on. We therefore have two choices, either cut emissions or face an existential crisis. Thin film devices such as solar cells and OLEDs offer a viable way to reduce our CO_2 emissions, either by providing low carbon electricity, or providing an efficient way to use the energy once generated.

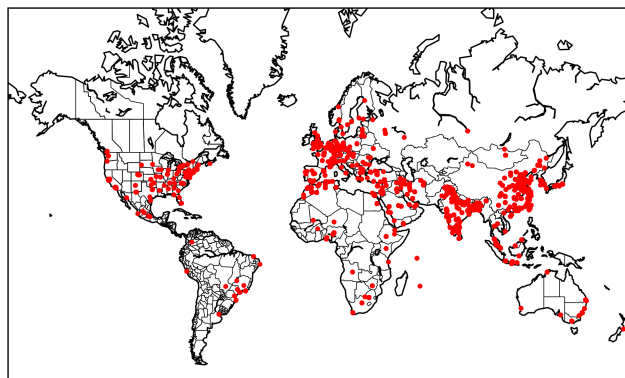


Figure 1.1: A map of locations where OghmaNano has been downloaded.

It is therefore important that technologies based on thin film devices continue to be developed and succeed. By developing and releasing OghmaNano, I hope, I am enabling scientists throughout the world to understand these devices a little bit better, which I hope will contribute in a very small way to solving our climate crisis.

Solar cells and OLEDs happen to come from a class of devices called diodes. This class of devices has many uses including optical sensors, medical sensors, switches, rectifiers. Thus as a pleasant side effect of OghmaNano the development of these devices is also being helped.

1.3 About this book/manual

This book is intended to be the definitive guide to simulating devices with OghmaNano. The idea is that one can read this book and learn in a step-by-step way how to simulate many modern opto-electronic devices with very limited prior knowledge. However, not all aspects of this book are yet finished. I therefore recommend you also watch the YouTube channel (and subscribe! ;)) where I describe many of the features in more detail and give demonstrations on the use of the model. I would suggest you treat the videos as lectures (and take notes) rather than entertaining videos (well I hope they are entertaining too!). New releases are generally announced on Twitter, which I also suggest you follow to make sure you are using an up-to-date version. I often release version every week, a version that is 6 months old is considered very old indeed. Please read papers which were published from this model - do also read the supplementary information (SI) to the papers, as I often write about the model in there. This book starts off with explaining how to simulate organic solar cells. This is because organic solar cells are the easiest class of device to simulate, it then moves onto Perovskite devices, and OLEDs. More complex classes of devices follow. If you are new to simulation work or in indeed OghmaNano, I suggest you start with the first chapters and work your way to the more complex devices.

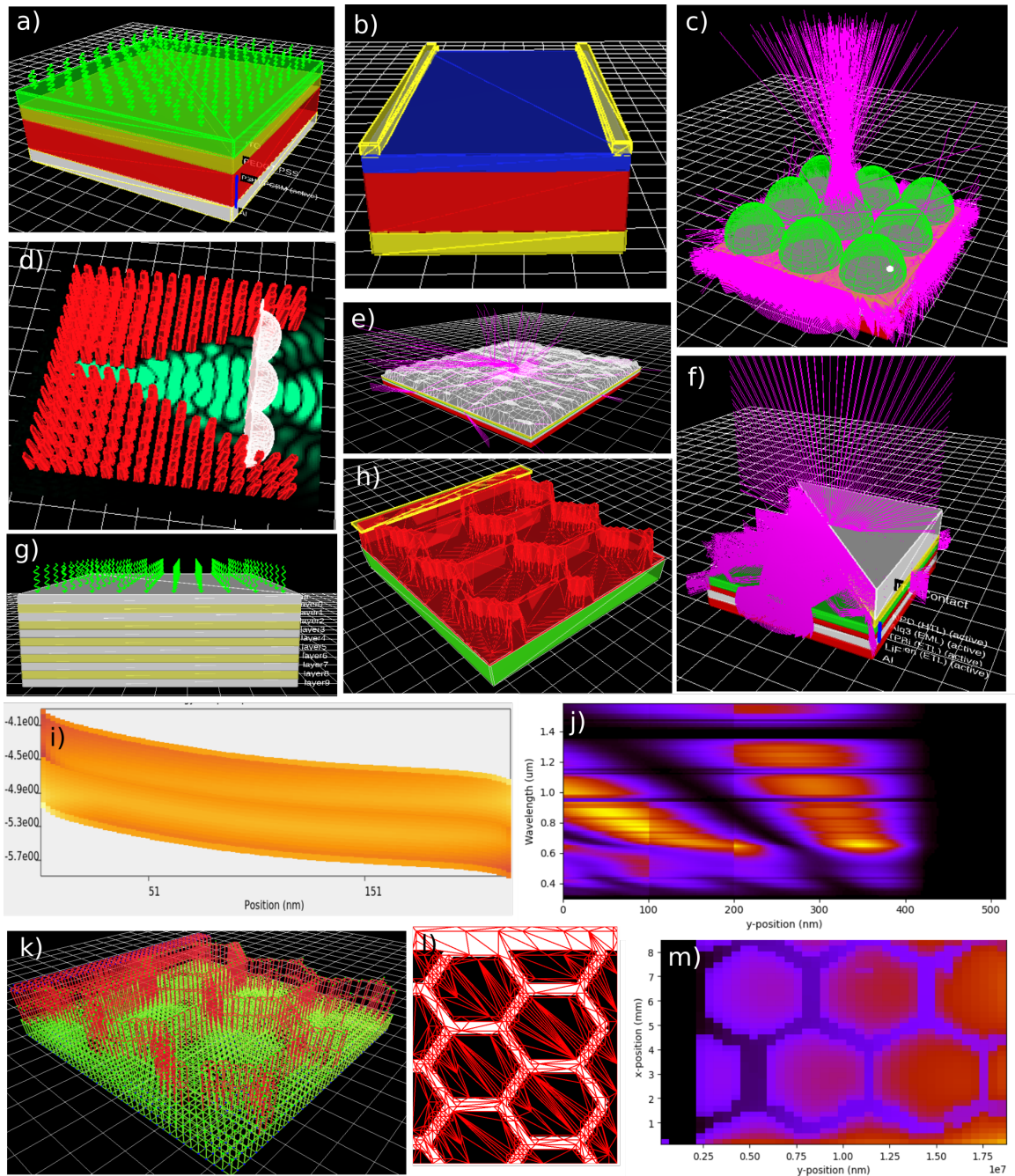


Figure 1.2: a); Organic/Perovskite solar cell simulation; b) OFET transistor simulation; c) Microlens simulation; d) Photonic waveguide simulation; e) light escaping structured films; f) OLED simulation; g) Optical filter simulation; h) large area device modelling (hexagonal contacts); i) Mapping carrier density in position/energy space; j) Building complex 3D meshes; and k) resistance maps of large area devices.

1.4 What is the history of OghmaNano?

I started writing OghmaNano just after finishing my PhD in 2009 while taking a break for academia and deciding what to do next. At that time it was a simple 1D drift-diffusion diode model designed to simulate solar cells which did not take account of disorder. Over the next 14 or so years the model has been significantly expanded to model many classes of material system and classes of devices. Since 2009 thousands of people have downloaded OghmaNano and hundreds (the list is by definition always out of date) of people have published their own papers using the model. If you publish with OghmaNano let me know and I will update the list to include your paper.

1.5 What is the roadmap for OghmaNano?

The aim is to make OghmaNano a completely general opto-electronic model which can be used by anyone to learn about and explore the world of novel opto-electronic devices. I want OghmaNano to be an engine which people can use to push their own research forward and for education. The exact road map on how to get there is not defined. As collaborators contact me asking for new features I add them, what comes first depends on what people want.. I never view OghmaNano as finished, and release improvements in small increments, therefore if you discover and report a bug, check back in a week or so to see if it is fixed in the next version. The same goes for this book, it evolves weekly as I write it. So if a section is missing, check back next week it might be finished.

1.6 Using OghmaNano in industrial/academic work

You are free to use OghmaNano in industrial/academic work. In fact, I'm happy if you do so. However, the following conditions apply:

1. If you use OghmaNano to generate results, then clearly say so in your work. This can be as simple as one sentences saying: "we used OghmaNano to perform the simulations"
2. If you publish a book, paper or thesis where OghmaNano has been used you must cite at least three papers associated with the model. To find out which papers to cite, click on the area indicated in red in figure 1.3 when using the model. PLEASE do not cite the manual. I can't include the manual in paper lists when applying for funding.

I ask you to do this because citations are an easy way to demonstrate that people are using OghmaNano. Demonstrating use is key to finding money/people to continue the development of OghmaNano. So by doing this you are guaranteeing the future of OghmaNano and its continued availability for others. Thank you!

1.7 Bugs

I get quite a lot of feature requests from people wanting features added or for bugs to be fixed. I really appreciate the feedback! However, I am currently employed at a UK University and my time is split between teaching, research and admin. My performance in my job is measured by the number of high impact papers I push out per year. I therefore have to prioritize feature requests and bug fixes for people who would like to write a paper with me (i.e. my collaborators).... Therefore if you would like:

1.7. BUGS

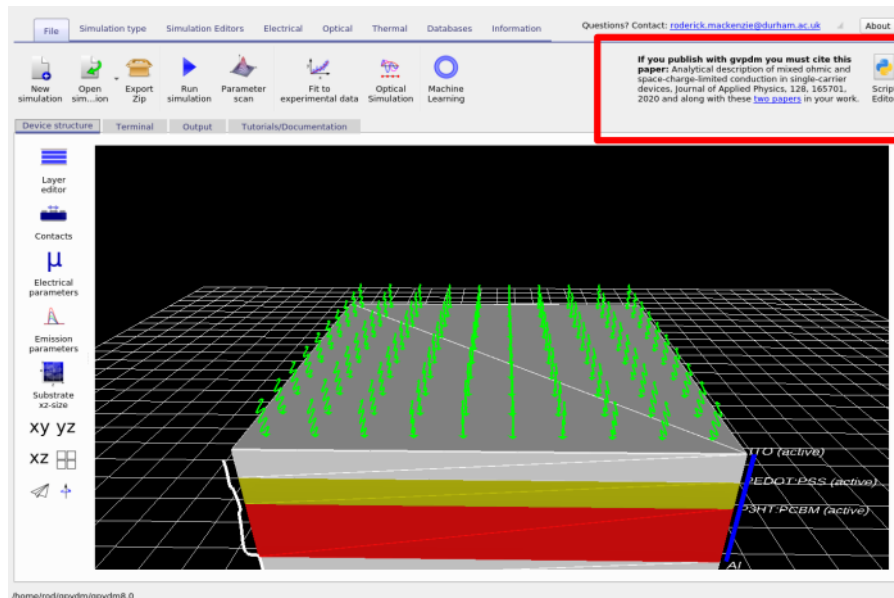


Figure 1.3: If you click on the area indicated by the red box, the model will tell you which papers should be cited.

- A bit of advice on how to do x or y with the model then please do feel free to shoot me a mail, and I will do my best to get back to you. If you don't hear back from me just send the mail again.. I get loads of e-mails, and things get lost if I don't answer quickly.
- If you want to report a bug, then please do that, and I will do my best to fix it in the next release. But I can't promise when it will be fixed.
- If you would like a features added or a steady stream of help (i.e. you are asking for my time) then please consider inviting me to join in your work and collaborate on a joint paper. I am happy to add whatever feature you want to the model, or fix what ever bug you may have but in return I would ask for the inclusion of my name on the author list. By doing this it makes it much easier for me to justify sinking time into your project.

If you don't need help from me to use OghmaNano then please feel free to do what you want with the results - no need to contact me, but do cite it correctly.

Chapter 2

Installing OghmaNano

2.1 Windows (if you have admin rights)

Go to the download page for OghmaNano at <http://www.Ogham-Nano.com/windows.php> and download the latest version. Simply double click on it and say yes to all questions, it will then install on your PC and an icon will appear in the start menu. I recommend you install it in the default directory.

In general I release a new version every couple of weeks and it's worth keeping your version up-to-date. On modern versions of windows, windows will ask you if you want to install an unsigned executable from an unknown author, and warn you that this could damage your computer. The reason you get this message is because I have not cryptographically signed the .exe file. I have not signed it because I do not own a private cryptographic key with which to do this. To get such a key I would have to send my passport off to a key authority to prove who I am and then pay them 500 pounds/year for the privilege of them validating who I am. Needless to say, that I am not very excited about paying 500 pounds/year so you will just have to click away the warnings from windows.

2.2 Windows (No admin rights)

If you don't have admin rights to your computer it can be hard to install new software, OghmaNano offers the option of running OghmaNano while not properly installed. Download the zip file containing OghmaNano from https://www.Oghma-Nano.com/download_no_admin.php. Once you have downloaded the zip archive, open the zip file and extract the folder *pub* to c:\. Then rename the folder to be called c:\OghmaNano. Once you have done this run the executable c:\OghamNano\OghmaNano.exe (see figure 2.1).

2.2. WINDOWS (NO ADMIN RIGHTS)

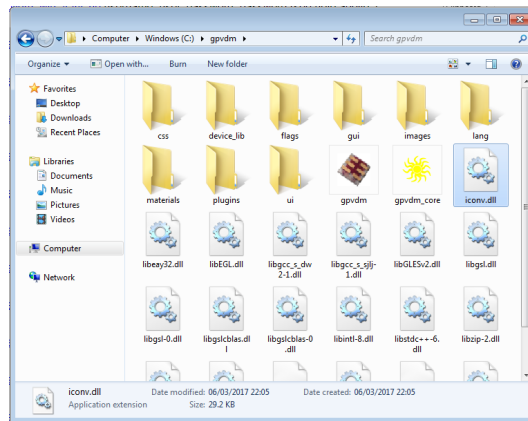


Figure 2.1: Running and installing OghmaNano. Double click on the OghmaNano icon to run the model.

Chapter 3

Getting started

3.1 Simulating a JV curve of a simple solar cell

No matter which type of device you want to simulate, if you are new to OghmaNano my advice is to start off with this organic solar cell simulation. Organic solar cells are by far the most simple class of device you can simulate, and will let you understand the basics of the package without having to deal with 2D effects, perovskite ions of light emission. This chapter will guide you through your first organic solar cell and explain the nuts and bolts of running simulations with OghmaNano. Once installed OghmaNano appear on the start menu, click on it to launch it. Once run, a window resembling that in figure 3.1 will appear.

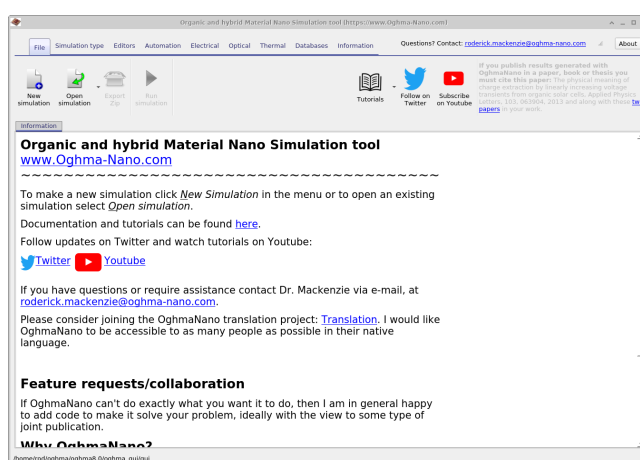


Figure 3.1: The main OghmaNano simulation window.

3.1.1 Making your first simulation

Click on the *new simulation* button. This will bring up the new simulation window (see figure 3.2). From this window double click on the *Organic Solar Cells* icon. This will bring up a sub menu of different types of Organic Solar cells (see figure 3.3). The majority of these device simulations have been published in papers and calibrated to real organic solar cells. The oldest is the (non-inverted) P3HT:PCBM device from 2012 [3] and the newest are the PM6:Y6 devices from 2022 [4, 5]. *Double click on the P3HT:PCBM simulation for this example and save the new simulation to disk.*

Once you have saved the simulation, the main OghmaNano simulation window will be brought up (see figure 3.4). You can look around the structure of the solar cell, by dragging the picture of the solar cell with your mouse. Try pressing on the buttons beneath the red square, they will change the orientation to the xy , yz or xz plane. Notice the x,y,z origin marker in the bottom left of the 3D window. The icon with four squares will give you an orthographic view of the solar cell.

Click on the button called *Run simulation*, to run the simulation (hint it looks like a blue play button and is located in the *file* one to the right of the "Simulation type ribbon"). The function key F9 will also run the simulation. On slower computers it could take a while. Once the simulation is done, click on the *Output* tab (see figure 3.1), there you will see a list of files the simulation has written to disk.

3.1. SIMULATING A JV CURVE OF A SIMPLE SOLAR CELL

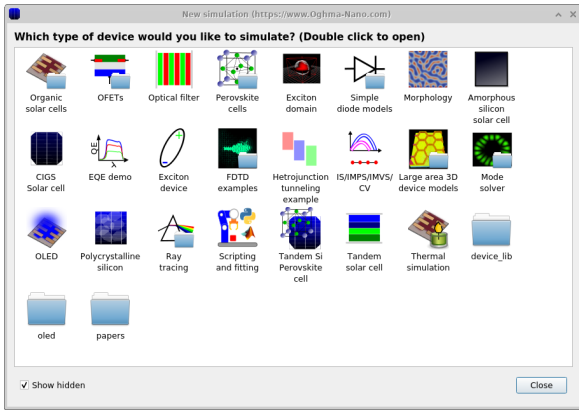


Figure 3.2: New simulation window, from here you can select different example simulations. It is often easier to start from a base simulation rather than build your own from scratch.

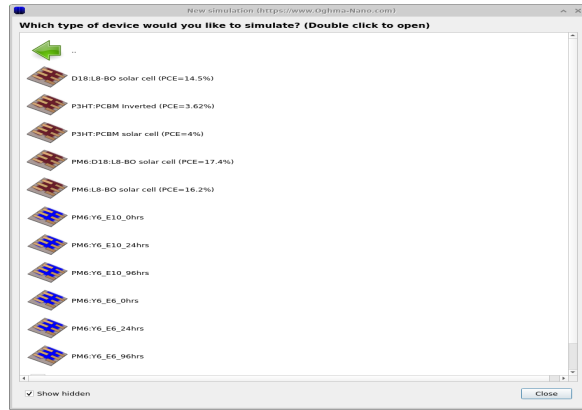


Figure 3.3: The organic solar cell sub menu. There are quite a few examples of organic solar cells in this menu. The majority of simulations have been used to produce papers [3, 4, 5].

What's the best place to save your simulation?

OghmaNano dumps a lot of data to disk, I therefore recommend you save the simulation to a local disk such as the C:\drive, a network drive or USB stick drive will be far too slow for the simulation to run. I would also not save the simulation onto OneDrive or Dropbox as they are also too slow and saving it there will generate a lot of network traffic. If you are a power user doing a lot of fitting of experimental data I would also recommend (at your own risk(!)) disabling any extra antivirus software you have installed, as quite often the antivirus software can't keep up with the read/writes to disk.

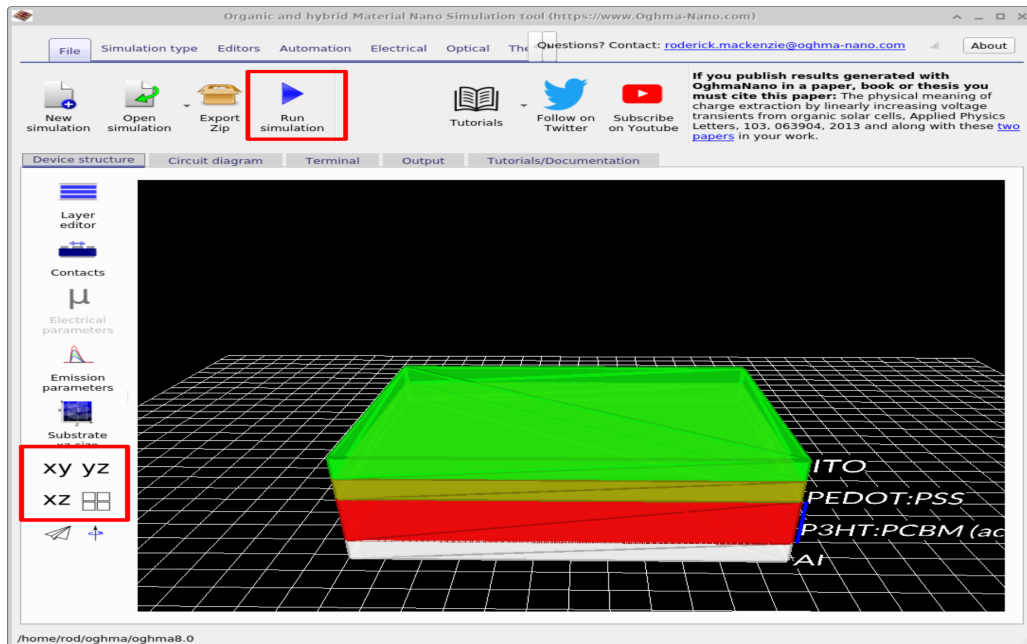


Figure 3.4: The main OghmaNano simulation window with the xy, yz and xz buttons visible. The play button is also visible which is used to run the simulation, the function key F9 can also be used to run the simulation.

3.1.2 The output from your first simulation

After you have clicked on the *Run Simulation* button (or pressed the function key F9) to run the simulation, the results from the simulation will have been written to disk. To view these results click on the *Output* tab in the main window. There you will see the output from the simulation, this is visible in figure 3.1

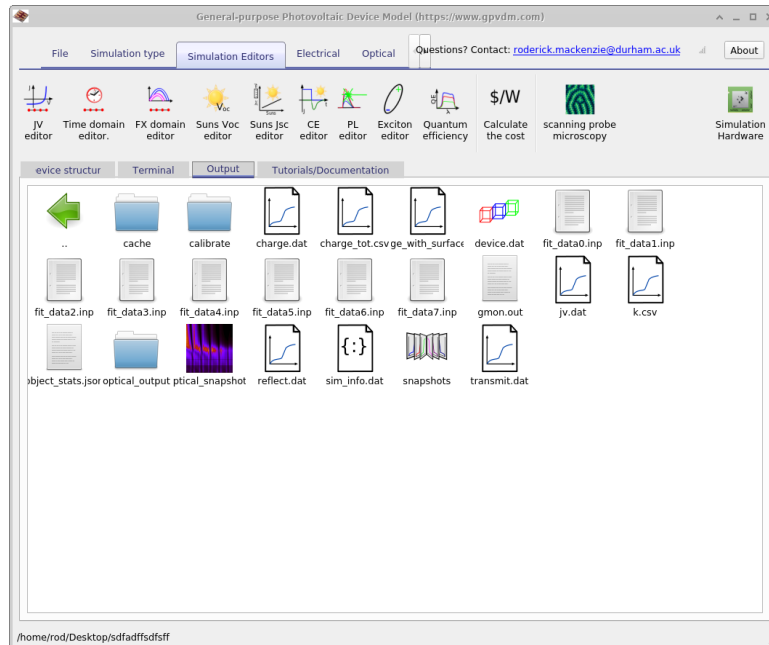


Figure 3.5: The *Output* tab this is just like windows file explorer, you can explore the simulation directory tree.

Key files the simulation produces are listed in the table below:

File name	Description
<i>jv.dat</i>	Current v.s. voltage curve
<i>charge.csv</i>	Voltage v.s. charge density curve
<i>device.dat</i>	The 3D device model
<i>fit_data * .inp</i>	Experimental data for this device.
<i>k.csv</i>	Voltage v.s. Recombination constant k
<i>reflect.csv</i>	Optical reflection from device
<i>transmit.csv</i>	Optical transition through device
<i>snapshots</i>	Electrical snapshots see 17.1
<i>optical_snapshots</i>	Optical snapshots see 17.3
<i>sim_info.dat</i>	Calculated V_{oc} , J_{sc} etc.. see 4.1.4
<i>cache</i>	Cache see 17.4

Table 3.1: Files produced by the JV simulation

Try opening *jv.dat*. This is a plot of the voltage applied to the solar cell against the current generated by the device. These curves are also sometimes called the *characteristic diode curve*, we can tell a lot about the solar cell's performance by looking at these curves. Hit the 'g' key to bring up a grid.

3.1. SIMULATING A JV CURVE OF A SIMPLE SOLAR CELL

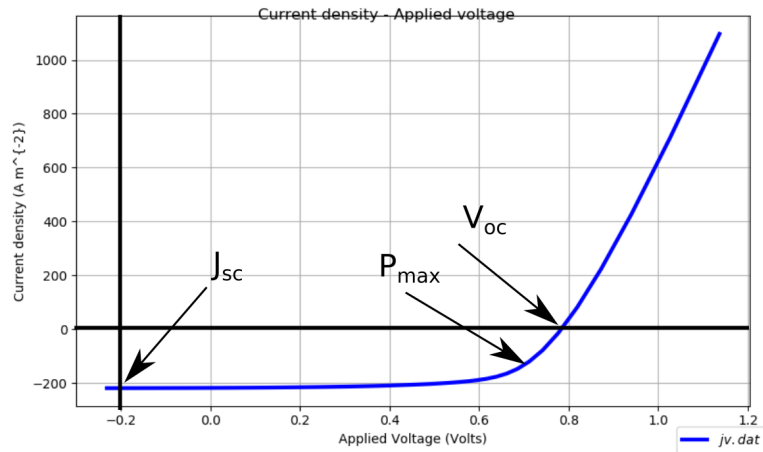
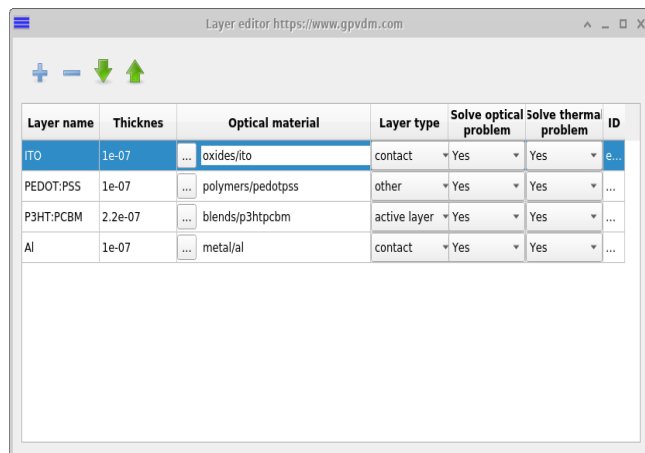


Figure 3.6: The output tab

Now try opening up the file *sim.info.dat*, this file displays information on the performance of the solar cell, such as the Open Circuit Voltage (V_{oc} - the maximum Voltage the solar cell can produce when illuminated), efficiency (η - the efficiency of the cell), and short circuit current (J_{sc} - the maximum current the cell can produce when it is illuminated). Figure 3.6, shows where you can find these values on the JV curve. The *sim.info.dat* file contains a lot of other parameters, these are described in detail in section 4.1.4.

Question 1: What is the J_{sc} , V_{oc} and Fill Factor (FF) of this solar cell? How do these number compare to a typical Silicon solar cell? (Use the internet to find typical values for a Silicon solar cell.)

3.1.3 Editing device layers



The screenshot shows a window titled "Layer editor https://www.gpvd.com" with a table of layers. The table has columns for Layer name, Thickness, Optical material, Layer type, Solve optical problem, Solve thermal problem, and ID. The layers listed are ITO, PEDOT:PSS, P3HT:PCBM, and Al.

Layer name	Thickness	Optical material	Layer type	Solve optical problem	Solve thermal problem	ID
ITO	1e-07	oxides/ito	contact	Yes	Yes	e...
PEDOT:PSS	1e-07	polymers/pedotpss	other	Yes	Yes	...
P3HT:PCBM	2.2e-07	blends/p3htpcbm	active layer	Yes	Yes	...
Al	1e-07	metal/al	contact	Yes	Yes	...

Figure 3.7: The layer editor window.

Any device in OghmaNano consists of a series of layers (this is sometimes referred to as the epitaxy - this is a term which comes from inorganic semiconductors). The layer editor can be accessed from the main simulation window, under the *device structure* tab. This is visible towards the top of figure 3.4, and the layer editor is visible in figure 3.7. Within the window is a table that describes the structure of the device. The column thickness describes the thickness of each layer. The P3HT:PCBM layer is the layer of material which converts photons into electrons and holes, this is commonly called the active layer.

An active layer thickness of 50nm is considered very thin for an organic solar cell, while an active layer of 400nm is considered very thick (too thick for efficient device operation). Vary the active layer between 50 nm and 400 nm, for each thickness record the device efficiency (I suggest you perform the simulation for at least eight active layer widths).

More on the layer editor

The layer editor has the following columns:

- Layer name: Is the English name describing the layer. You can call your layers what you want (i.e. ITO, PEDOT, fred or bob) it has no physical meaning.
- Thickness: Is the layer thickness given in meters.
- Optical material: Specifies the n/k data which is used to describe the materials optical properties. In the simulation the n/k data are taken from experimental values stored in the optical database 14.1 and have nothing to do with the electrical material properties such as effective band gap.
- Layer type: Specifies to the simulation how the layer is treated when performing a simulation. There are three types of layer
 - active: This type of layer is electrically active and the drift diffusion solver will solve the electrical equations in this layer type. See section 19.2. You can have as many active layers as you like but they must be contiguous.
 - contact: This tells the model that a layer is a contact and a voltage should be applied, see section 3.1.8 for more details.
 - other: Any layer which is not a contact or active.

Which layers should be active?

A common mistake people make when starting to simulate devices is to try to make all the layers in their device active because their logic is: Current must be flowing through them so they must be active right? However, in for example a solar cell only the BHJ or in a perovskite device the perovskite layer will have both species of carriers (electrons+holes) and complex effects such as photogeneration, recombination and carrier trapping. So in this layer it makes sense to

3.1. SIMULATING A JV CURVE OF A SIMPLE SOLAR CELL

solver the drift diffusion equations. Other layers which don't have both species of carriers can be treated simple parasitic resistances see section 3.1.6. I would only recommend setting other layers of the device to active (such as the HTL/ETL) if you are trying to investigate effects such as s-shaped JV curves or devices which clearly need multiple active layers such as OLEDs. In general, try to minimize the number of active layers and always keep simulations as simple as possible to explain the physical effects you see.

Task 2: Plot a graph (using excel or any other graphing tool), of device efficiency v.s. thickness of the active layer. What is the optimum efficiency/thickness of the active layer? Also plot graph V_{oc} , J_{sc} and FF as a function of active layer thickness. J_{sc} is generally speaking the maximum current a solar cell can generate, try to explain your graph of J_{sc} v.s. thickness, [Hint, the next section may help you answer this part of the question.]

3.1.4 How do solar cells absorb light?

In this section we are going to learn how a solar cells interact with light. Firstly, let's have a look at the solar spectrum. Sunlight contains many wavelengths of light, from ultraviolet light, though to visible light to infrared. The human eye can only see a small fraction of the light emitted by the sun. OghmaNano stores a copy of the suns spectrum to perform the simulations. Let's have a look at this spectrum, to do this go to the *Database* tab, the choose *Optical database*. This should, bring up a window as shown in figure 3.8

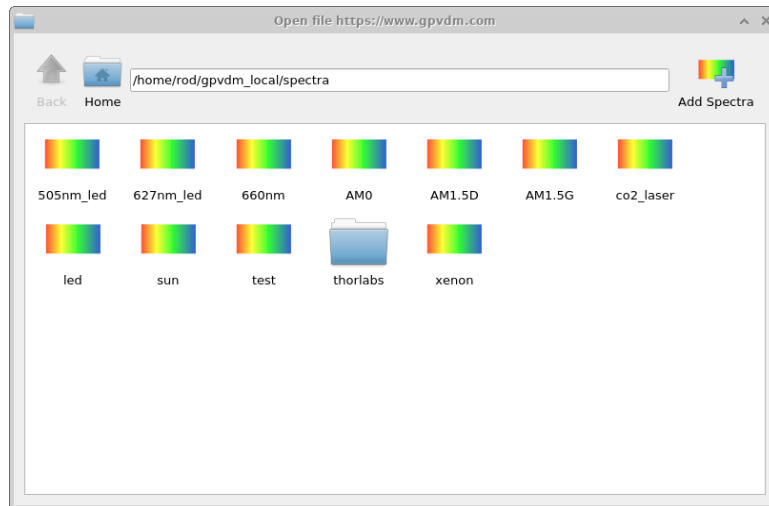


Figure 3.8: The optical database viewer

Double click on the icon called, *AM1.5G*, this should bring up a spectrum of the sun's spectrum. Have a look at where the peak of the spectrum is. Now close this window, and open the spectrum called *led*. Where is the peak of this spectrum.

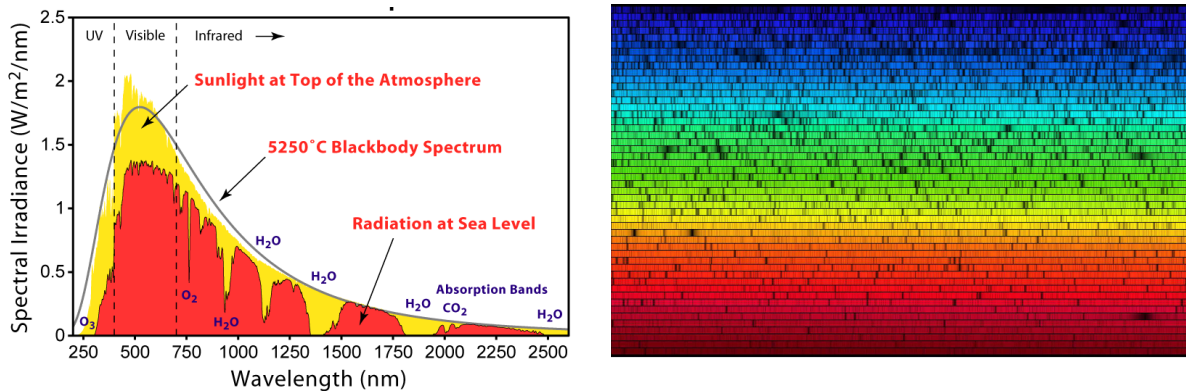


Figure 3.9: a: A plot of the entire solar spectrum. b: The image below shows the solar spectrum at 392 nm (blue) to 692 nm (red) as observed with the Fourier Transform Spectrograph at Kitt Peak National Observatory in 1981. R. Kurucz

Question 3: Describe the main differences between the light which comes from the LED and the sun. Rather than referring to the various regions of the spectrum by their wavelengths, refer to them using English words, such as *infrared*, *UltraViolet*, *Red*, and *Green* etc... you will find which wavelengths match to each color on the internet. If you were designing a material for a solar cell, what wavelengths would.

3.1. SIMULATING A JV CURVE OF A SIMPLE SOLAR CELL

3.1.5 Light inside solar cells

As you will have seen from when you first opened the simulation, the solar cells are often made from many layers of different materials. Some of these materials, are designed to absorb light, some are designed to conduct charge carriers out of the cell. The simulator has a database of these materials, to look at the database, click on the *Database* tab, then click on *Material database*. This should bring up a window as shown in figure 3.10, once this is open navigate to the directory *polymers*, and double click on the material *p3ht*, in the new window click on the tab *Absorption* (see figure 3.11). This plot shows how light is absorbed in the material as a function of wavelength.

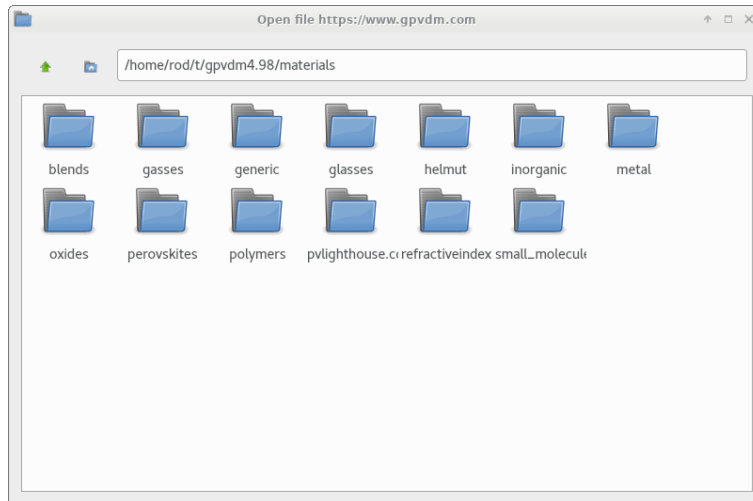


Figure 3.10: The materials database

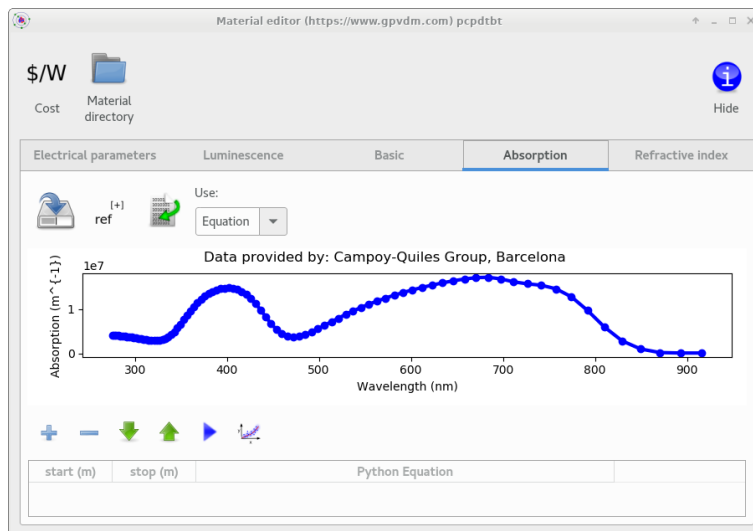


Figure 3.11: Optical absorption of the light.

Question 4: What color of light does the polymer *p3ht* absorb best? Which material in the *polymers* directory do you think will absorb the sun's light best?

3.1.6 Parasitic elements

Many devices have parasitic shunt and series resistances associated with them. Shunt resistances (R_s) are caused by conduction straight through the device in thin novel devices this is often caused by impurities in the material system. Parasitic series resistances (R_s) are often associated with the resistance of the contacts, the resistance of the HTL/ETL or any other resistances which are not associated with the active layer. These resistance can be seen for a typical solar cell in figure 3.12 also shown in the figure is the ideal diode of the device. These resistances can be set in the parasitic component window shown in figure 3.13

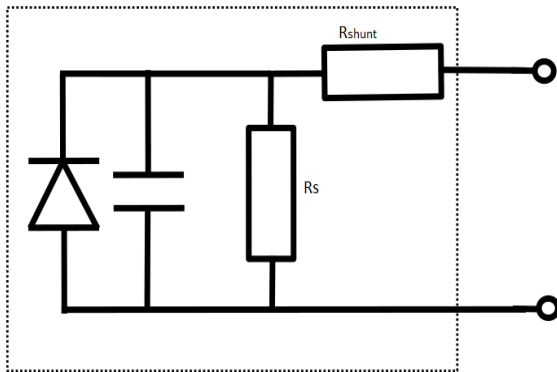


Figure 3.12: Circuit model of a solar cell.

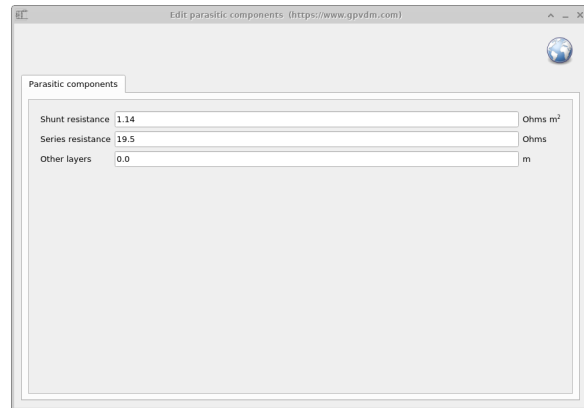


Figure 3.13: The parasitic component editor.

You can change the values of series and shunt resistance in OghmaNano, by going to the *Electrical* tab and then clicking on the *Parasitic components* button. Due to the flat broad contacts on a solar cell, there is often a capacitance associated with the device, this is important for transient measurements and can be calculated with the equation:

$$C = \frac{\epsilon_r \epsilon_0 A}{d + \Delta} \quad (3.1)$$

where A is the area of the device ϵ are the hyperactivities, and d is the thickness of the device. Often for various reasons the measured capacitance of the device does not match what one would expect from the above equation. Therefore the term "Other layers" (Δ) has been added to the parasitic window to account for differences between measured capacitance and layer measured layer thicknesses.

Task 5: In the optical tab you will find a control called *Light intensity*, this controls the amount of light which falls on the device in Suns. Set it to zero so that the device is in the dark. Then run two JV curve simulations, one with a shunt resistance of 1 Ohm m^2 and one with a shunt resistance of $1 \times 10^6 \text{ Ohm m}^2$ (Hint you will have to enter $1e6$ in the text box). What happens to the dark JV curve? Now try running the same same simulations again but in the light.

3.1. SIMULATING A JV CURVE OF A SIMPLE SOLAR CELL

3.1.7 Solar cells in the dark

So far, all the simulations we have run have been performed in the light. This is a logical, as usually we are interested in solar cell performance only in the light. However, a lot of interesting information can be gained about solar cells by studying their performance in the dark. We are now going to turn off the light in the simulation. From the *Optical* tab set the *Light intensity (suns)* drop down menu to 0.0 Suns, this can be seen in figure 3.14. The photons in the 3D image should disappear as seen in figure 3.14.

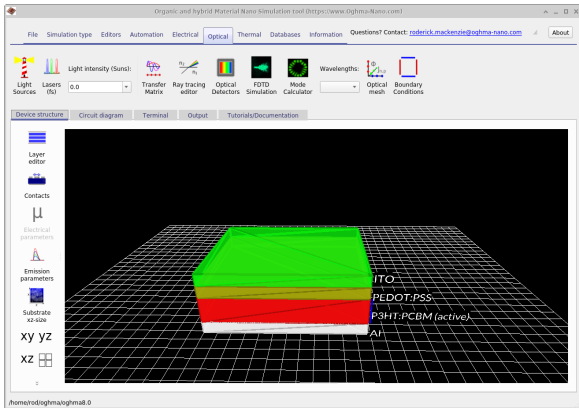


Figure 3.14: Running OghmaNano in the dark, the Light intensity drop-down menu has been set to 0 Suns and the photons have disappeared from the image.

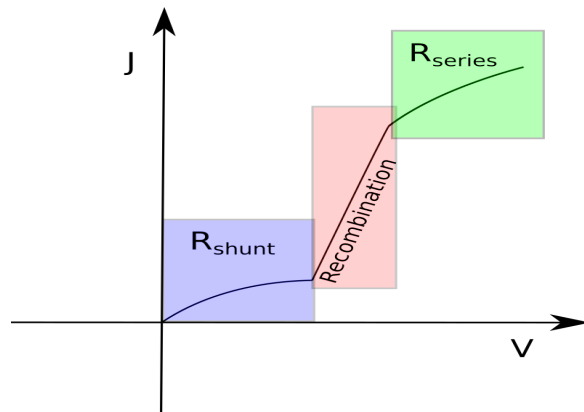


Figure 3.15: A sketch of a typical dark JV curve.

Now set the shunt resistance to $1M\Omega m^2$, and run a simulation. Plot the jv curve. It is customary to plot jv curves on a x-linear y-log scale. To do this in the plot window, hit the 'l' key to do this. The shape should resemble, the JV curve in figure 3.15. Certain solar cell parameters affect different parts of the dark JV curve differently, the lower region is affected very strongly by shunt resistance, the middle part is affected strongly by recombination, and the upper part is strongly affected by the series resistance.

Question 6: What values of series and shunt resistance, would produce the best possible solar cell? Enter these values into the device simulator and copy and paste the dark JV curve into your report.

3.1.8 The contact editor

The contact editor is used to configure the electrical contacts. Which layers act as contacts is configured in the layer editor see section 3.1.3. The contact editor has the following fields:



Figure 3.16: The contact editor

- Name: The name of the contact, this can be any English word. It has no physical meaning.
- Top/Bottom: Sets if the contact is on the top, bottom or in 2D simulation left and right of the device are also valid.
- Applied voltage: Sets the applied voltage on the contact. You first have to select what type of applied voltage you want:
 - Ground: This will set the contact to zero volts i.e. ground. 0V is always taken as ground.
 - Constant bias: This will apply a constant bias to a contact. It can be set to zero, and would then be equivalent to ground. In OFET simulations the voltage value can be set to bias one contact to a desired constant voltage.
 - Change: If a contact is set to 'Change' this tells the simulation to apply a changing voltage to this contact. For example if you are performing a JV sweep, the sweep voltage will be applied to this contact. Similarly if you are doing an IS simulation (TPV, TPC, ToF etc..) the voltage will be applied/measured to this contact.
- Charge density: This sets the majority charge density on the contacts. The Fermi-offset is calculated from the charge density. The model does not use Fermi-offset as an input, it uses charge density.
- Majority carrier: This sets the majority carrier density to electrons or holes.
- Physical model: This selects if you have ohmic contacts or schottky contacts. I recommend using ohmic contacts.

Task 7: For a good contact which results in a high efficiency device, the Fermi-offset will be exactly 0 eV or very small. Firstly set the Fermi-offset to zero for both contacts, and run a simulation. What efficiency cell do you get? Now set the Fermi-offset to 0.3eV what efficiency cell do you now have? Make a note of the charge densities on the contacts which these Fermi-offsets produce.

3.1.9 Electrical parameters

The electrical parameter editor enables you to change the electrical parameters associated with the active layers. Here you can change mobilities, trap constants etc. If you set a layer to active within the layer editor it will appear within the electrical parameter editor. The toolbar at the top of the window allows you to turn off and on various electrical mechanisms including:

- Drift diffusion: This enables drift diffusion within the layer. In most circumstances if a layer is set to be active there is no reason why you would want to turn this option off. The one example is in the insulating layer of an OFET.
- Auger recombination: This switches on and off Auger recombination. See 8.4.4 for more information.
- Dynamic SRH traps: This is used to turn on and off dynamic SRH traps. See section 8.4 for more information. This option should be turned on when modeling disordered semiconductors such as organic materials.
- Equilibrium SRH traps: This can be used to introduce a single equilibrium trap level. See section 8.4 for more information.
- Excitons: This enables the exciton diffusion equation to be solved along with the electrical equations. See section 12.2 for more information.
- Excitons: This enables singlet and triplet states to be modelled.

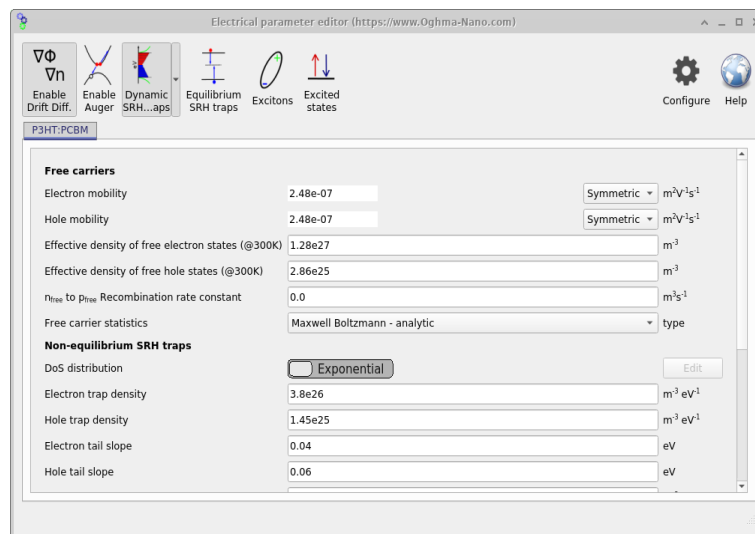


Figure 3.17: Electrical parameter window

Task 8: The values of electron mobility dictate how easily charge can move in the device. You can think of this value as akin to resistance or a sort of microscopic resistance. Try increasing the mobilities by two orders of magnitude and look what happens to the light JV curve of the device and the efficiency, FF, V_{oc} and J_{sc} . Do you think it is good to have a low or high value of mobility?

Task 9: Recombination is described later in detail but for now we can simply think of it as how many electrons and holes meet each other in a given time. As stated above there are various types of recombination which can happen in organic semiconductors, but for now we will *just consider* the case when a free electron meets a free hole. This is sometimes called bi-molecular recombination, the equation for this is given by:

$$R(x) = kn(x)p(x) \quad (3.2)$$

Where $n(x)$ is the density of electrons and $p(x)$ is the density of holes, and k is a rate constant. Before trying to understand this rate, firstly turn off the more complex SRH recombination by clicking on the *Dynamic SRH traps* in figure 3.17. You will notice lots of text boxes disappear. Then try changing the value of k which is set in the text box called n_{free} to p_{free} Recombination rate constant, from $1e-15$ to $1e-20$ in five steps. Run a simulation each time you change the value and make a graph of the efficiency of the cell as you change the value.

How do I know what electrical parameters to use?

For traditional semiconductors that have been studied for years such as AlGaAs or InP the values of charge carrier mobility, band gap, electron affinity (etc..) are well known and can simply be looked up on sites such as this or in books such in Piprek's [6] excellent book. These materials are highly pure (99.99999999%) (the so-called "eleven nines" purity). This means that when one has a sample of such a semiconductor one knows exactly what one has in the hand and what its physical properties will be. Organic semiconductors (also other novel materials such as perovskites etc..) on the are typically only 99.9% on a good day, that is a whole eight orders of magnitude less pure than their traditional counterparts. This means that when one has a sample of such a material one is not exactly sure what material one has hold of so it's harder to know what the values of mobility etc will be.

Furthermore, traditional semiconductors are very ordered, this means that the atoms within them pack in a regular lattice (think marbles packing in a biscuit tin) this again helps make their electronic properties predictable. Novel semiconductors on the other hand are typically much more disordered than their traditional counterparts and consist of a higgldy piggldy collection of polymers/molecules (or perovskite domains etc..), and the exact structure of these materials depends very much on how they were deposited. This means that due to fabrication techniques/conditions varying between different labs, nominally the same material produced by the same supplier but can behave very differently depending on when/who/where it was deposited by.

So this brings us back to the question that started this section, what parameters should I use for my novel device? Here are some tips:

- Use the base simulations provided in OghmaNano, these simulations have either been calibrated against real experimental devices or use very reasonable electrical parameters.
- Look in the literature and try to get an idea of what values are sensible ranges for the material systems you are looking at.
- Find some experimental data and make sure the current voltage curves produced by the model are within the same ball park as what you would expect experimentally, if they are totally out then you might need to tweak your electrical paramters.
- Fit the model to an experimental data set as was done in [3] and described in section 15 (This is however quite a hard thing to do though and not really recommended).

Chapter 4

Simulation modes and simulation editors

OghmaNano uses a modular architecture that enables the core solver to perform a variety of simulation types using . For example there is a plugin to perform steady state JV simulations, another plugin to perform frequency domain simulations, and another to calculate the Quantum Efficiency. They all leverage the same OghmaNano core solver but run it in a slightly different way with custom inputs and outputs. A list of the plugins and what they do can be found below:

- Plugins for various types of experiment
 - `juv`: To calculate steady state JV curves.
 - `suns_jsc`: Simulate suns v.s. Jsc curves.
 - `suns_voc`: Suns v.s. Voc simulations.
 - `eqe`: Simulates EQE.
 - `cv`: Capacitance voltage simulations.
 - `ce`: To simulate charge extraction experiments.
 - `time_domain`: A time domain solver for transient simulations.
 - `fx_domain`: Simulate the frequency domain response of a device, both electrical and optical excitation.
 - `pl_ss`: Calculate the PL spectrum at in steady state.
 - `mode`: Used to solve optical modes in 1/2D waveguides.
 - `spm`: Simulates scanning probe microscopy in 3D electrical simulations.
 - `equilibrium`: Equilibrium electrical simulations.
 - `exciton`: Exciton simulations.
 - `mesh_gen`: Generates meshes.
- Optical solver plugins
 - `fdtd`: Finite Difference Time Domain (FDTD) optical solver.
 - `optics`: Optical transfer matrix solver for 1D structures
 - `light_full`: Optical transfer matrix solver.
 - `light_qe`: Calculates optical profile using the experimental quantum efficiency.
 - `light_exp`: Calculates optical profile assuming exponential propagation of light in 1D structures.
 - `light_flat`: Calculates optical profile assuming flat optical profiles in the structure.
 - `light_constant`: Assumes user given values of generation rate in optical structures.
 - `light_fromfile`: Takes a generation rate from a file.

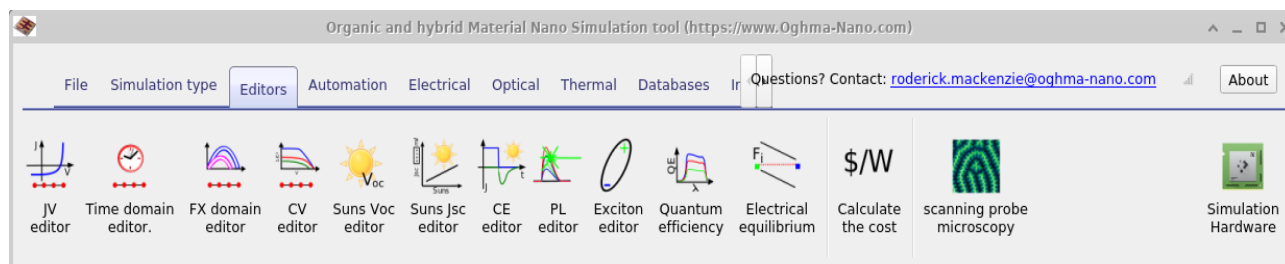


Figure 4.1: Simulation editors use this toolbar to edit the various simulation conditions your device will experience.

In the simulation editors ribbon (see Figure 4.1) you can see icons that represent each plugin, these are the simulation editors. By clicking on an icon in this ribbon you will be able to edit how the plugin performs the various simulations. For example in the JV simulation editor one can change the start/stop voltages of a voltage sweep. The JV editor can be seen in Figure 4.2. Within each simulation editor the user can define multiple so called *experiments*. This can be seen in below in Figure 4.2 and Figure 4.3, where two JV scans have been defined within the JV editor, one called *JV curve - low voltage* and another called *JV curve - high voltage*. One has a start voltage of 0.02V and stop voltage of 1.0V, while the other has a start voltage of 1.0V and a stop voltage of 10V. This feature is most useful in more complex experiments such as in time domain experiments where one may want to simulate multiple different voltage/light ramps/pulses for one device. There is no limit to how many *experiments* can be defined for each plugin.

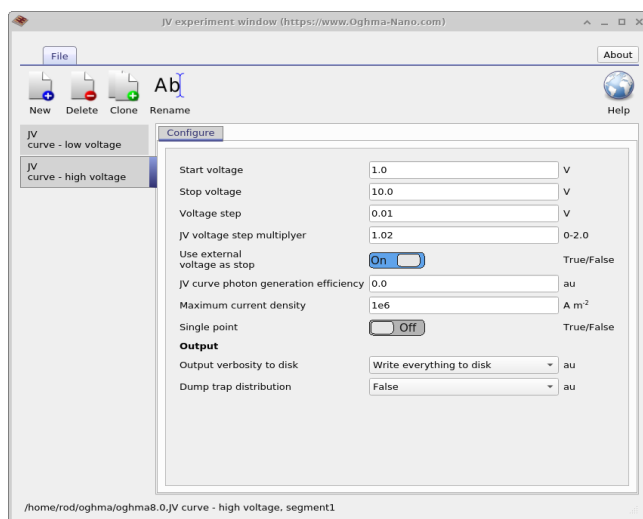


Figure 4.2: An experiment set up in the JV window for high voltage simulations.

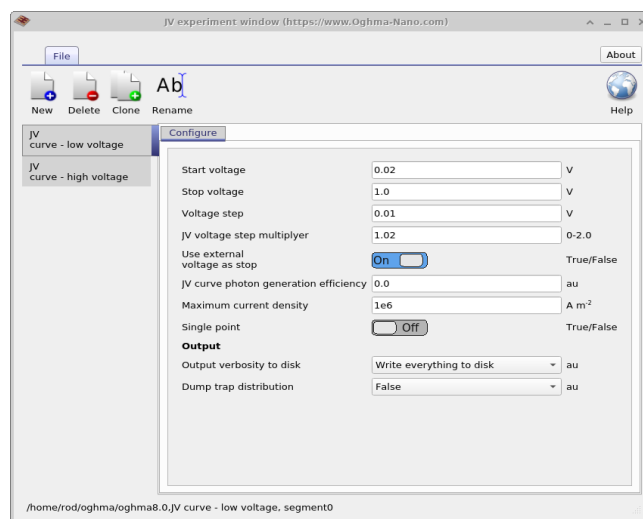


Figure 4.3: An experiment set up in the JV window for low voltage simulations.

Once an *experiment* has been defined an icon representing it will appear in the simulation mode ribbon shown in figure 4.4. You can see in the figure an icon for *JV curve low voltage* and *JV curve high voltage* that were defined in Figure 4.2 and 4.3. You can see in Figure 4.4 that *JV curve low voltage* is depressed. This means that when the simulation is run this simulation mode will be executed. If you select another simulation mode, then when the play button (or F9) is pressed that simulation mode will be run. Only one simulation mode can be run at a time.

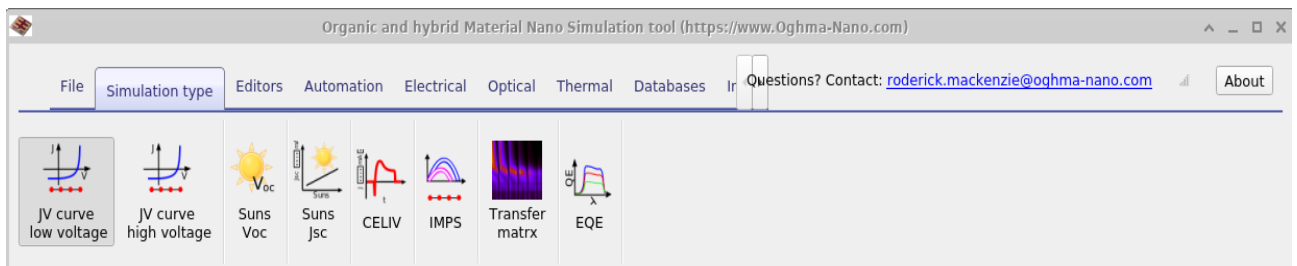


Figure 4.4: Selecting a simulation mode, in this case the *JV curve low voltage* has been selected so that when the user presses play that simulation mode will be run.

4.1 JV editor (Steady state simulation editor)

If you click on the JV editor icon in figure 4.5, the JV editor window will open shown below in figure 4.6.

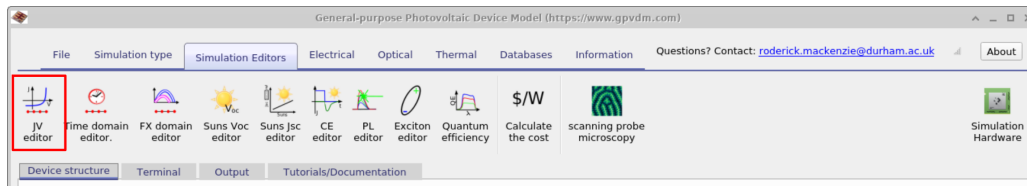


Figure 4.5: Opening the JV editor from the simulation editor ribbon.

4.1.1 Inputs

This window can be used to configure steady state simulations. It does not matter if you are running a current-voltage sweep on a solar cell or an OFET. This plugin will steadily ramp the voltage from a start voltage to a stop voltage. The voltage will be applied to the contact which has been set to *Change* in the contact editor (see section 3.1.8). You can set the start voltage, stop voltage and step size. Use *JV voltage step multiplier* to make the voltage step grow each step. The default is 1.0, i.e. no growth. It can be helpful to set the step multiplier to a value larger than 1.0 if you want to speed up the simulation but it should not be increased past about 1.05 or the simulation may struggle to converge.

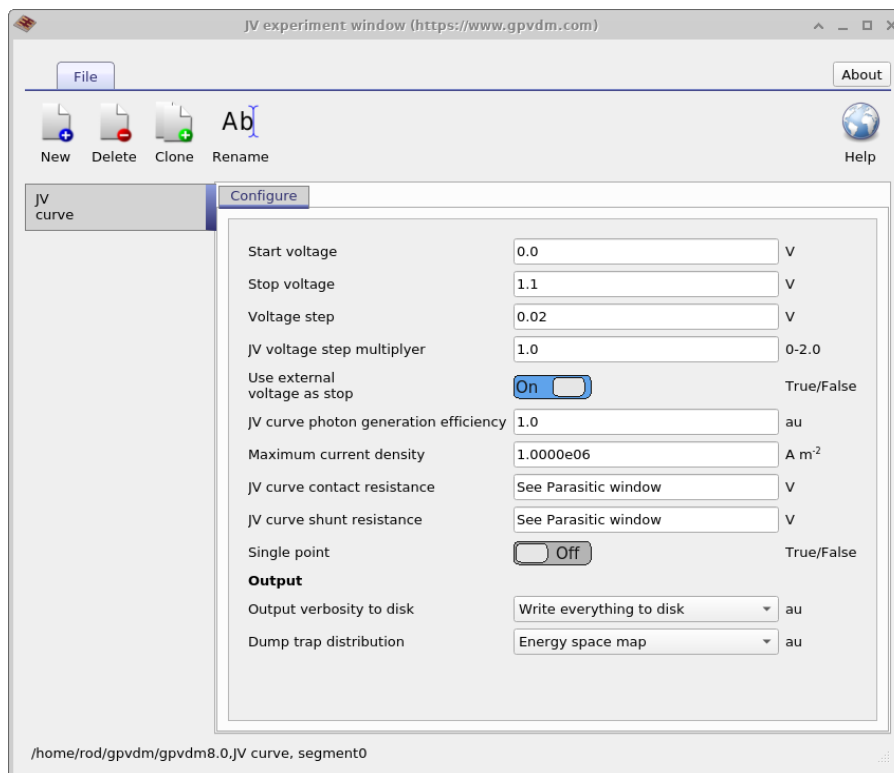


Figure 4.6: The JV editor editor window, use this to configure steady state simulations.

4.1.2 Outputs

The files produced by the JV simulation mode are given in table 4.1. As well as these files, by default OghmaNano will also write all internal simulation parameters to disk in the snapshots

4.1. JV EDITOR (STEADY STATE SIMULATION EDITOR)

directory. This includes band structure, potential, carrier distributions, generation rates etc.. this equates to about 50 files per voltage step. You can read more about this in the simulation snapshots section, see 17.1. This can considerably slow down the simulation, the user can therefore decide how much is written to disk by using the *Output verbosity to disk option* this can be set to; *Key results*, which will result in only key files being written to disk; *Nothing*, which will result in no results being written to disk; *Write everything to disk* which will result in a all possible information being written to disk and *Write everything to disk every nth step*, which will only out comprehensive internal simulation write data every nth step.

File name	Description
<i>ju.dat</i>	Current voltage curve
<i>charge.dat</i>	voltage charge density
<i>k.csv</i>	Recombination constant k
<i>sim_info.dat</i>	Calculated V_{oc} , J_{sc} etc.. see 4.1.4

Table 4.1: Files produced by the JV simulation

4.1.3 sim_info.dat

This is a json file containing all key simulation metrics such as J_{sc} , V_{oc} , and example sim_info.dat file is given below:

4.1.4 Steady state electrical simulation

In steady state electrical simulations such as performing a JV scan the sim_info.dat outputs the following parameters.

Symbol	JSON token	Meaning	Units	Equ.	Ref
FF	ff	Fill factor	au		
PCE	pce	PCE	percent		
P_{max}	P_{max}	Power at Pmax			
V_{oc}	V_{oc}	V_{oc}			
v_{ocR}	v_{ocR}	Recombination rate at P_{max}			
jv_{voc}	jv_{voc}				
jv_{pmax}	jv_{pmax}				
$v_{oc_{nt}}$	$v_{oc_{nt}}$	Trapped electron carrier density at V_{oc}			
$v_{oc_{pt}}$	$v_{oc_{pt}}$	Trapped hole carrier density at V_{oc}			
$v_{oc_{nf}}$	$v_{oc_{nf}}$	Free electron carrier density at V_{oc}			
$v_{oc_{pf}}$	$v_{oc_{pf}}$	Free hole carrier density at V_{oc}			
J_{sc}	J_{sc}	J_{sc}	Am^{-2}		
jv_{jsc}	jv_{jsc}	Average charge density at J_{sc}	m^{-3}		
jv_{vbi}	jv_{vbi}	Built in voltage	V		
jv_{gen}	jv_{gen}	Average generation rate			
$v_{oc_{np}}$	$v_{oc_{np}}$				
j_{pmax}	j_{pmax}	Current at P_{max}	Am^{-2}		
v_{pmax}	v_{pmax}	Voltage at P_{max}	V		

Symbol	JSON token	Meaning	Units	Equ.	Ref
μ_{jsc}^{geom}	μ_{jsc}	Avg. mobility at J_{sc}	$m^2V^{-1}s^{-1}$		
$\mu_{jsc}^{geom_micro}$	μ_{jsc}	Geom. avg. mobility @ J_{sc}	$m^2V^{-1}s^{-1}$		
	μ_{jsc}	Geom. avg. mobility @ J_{sc}	$m^2V^{-1}s^{-1}$		
	μ_{voc}	Average mobility @ V_{oc}	$m^2V^{-1}s^{-1}$		
μ_{voc}^{geom}	μ_{voc}	Geom. avg. mobility @ V_{oc}	$m^2V^{-1}s^{-1}$	$\sqrt{\langle \mu_e \rangle \langle \mu_h \rangle}$	
$\mu_{voc}^{geom_avg}$	μ_{voc}	Geom. avg. mobility @ V_{oc}	$m^2V^{-1}s^{-1}$	$\sqrt{\langle \mu_e \mu_h \rangle}$	
μ_{pmax}^e	μ_{pmax}	Avg. electron mobility @ P_{max}	$m^2V^{-1}s^{-1}$		
μ_{pmax}^h	μ_{pmax}	Avg. hole mobility @ P_{max}	$m^2V^{-1}s^{-1}$		
μ_{pmax}^{geom}	μ_{pmax}	Geom. avg. mobility @ P_{max}	$m^2V^{-1}s^{-1}$	$\sqrt{\langle \mu_e \rangle \langle \mu_h \rangle}$	
$\mu_{pmax}^{geom_micro}$	μ_{pmax}	Geom. avg. mobility @ P_{max}	$m^2V^{-1}s^{-1}$	$\sqrt{\langle \mu_e \mu_h \rangle}$	
μ_{pmax}^i	μ_{pmax}	Avg. mobility @ P_{max}	$m^2V^{-1}s^{-1}$		

4.1. JV EDITOR (STEADY STATE SIMULATION EDITOR)

Symbol	JSON token	Meaning	Units	Equ.	Ref
τ_{voc}	<i>tau_voc</i>	Recom. time at V_{oc}	s	$R = (n - n0)/\tau$	[7]
τ_{pmax}	<i>tau_pmax</i>	Recom. time at P_{max}	s	$R = (n - n0)/\tau$	[7]
τ_{voc}^{all}	<i>tau_all_voc</i>	Recomb. time at V_{oc}	s	$R = (n)/\tau$	[7]
τ_{pmax}^{all}	<i>tau_all_pmax</i>	Recomb. time at P_{max}	s	$R = (n)/\tau$	[7]
θ_{srrh}	<i>theta_srrh</i>	θ_{srrh} Collection coefficient at P_{max}	au		p.100 5.2a [8],[9]
θ_{srrh}	<i>theta_srrh</i>	θ_{srrh} Collection coefficient at P_{max}	au		p.100 5.2a [8],[9]

4.2 Time domain editor

Related YouTube videos:



Simulating optoelectronic sensors made from polymers.

The time domain editor can be used to configure time domain simulations, this is shown in Figure 4.7. You can see, as described in the previous section that one simulation editor can be used to edit multiple *experiments*. The panel on the left shows the editor being used to edit a CELIV simulation while the panel on the right shows the editor being used to edit a TPC simulation. The new, delete and clone buttons in the top of the window can be used to make new simulation modes. The table in the bottom of the window can be used to setup the time domain mesh, apply voltages or light pulses.

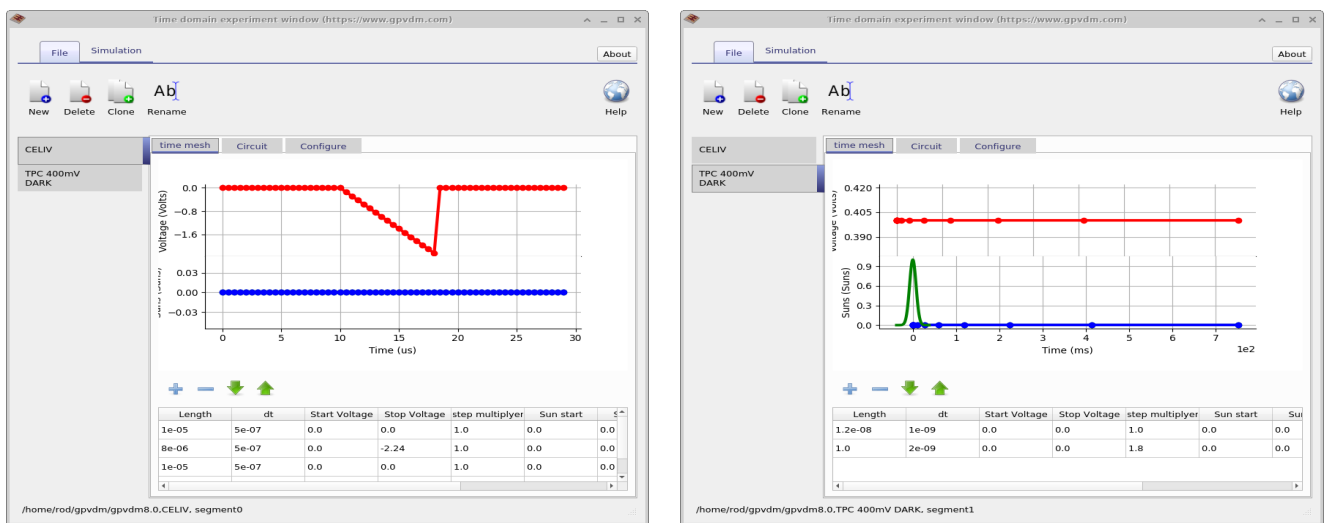


Figure 4.7: The time domain editor showing the user editing the duration of light/voltage pulses.

Figure 4.8 shows different tabs in of the time domain editor. The image on the left shows the circuit diagram used to model the CELIV experiment. The diode on the left represents the drift diffusion simulation while the other components represent various parasitic components. After the diode from the left next comes a capacitor used to model the charge on the plates of the device, then a shunt resistance and then the series resistance. The final resistor on the right represents the external resistance of the measuring equipment, this is by default set to zero but worth checking. The drop down menu on the top left of the image above the circuit diagram says *load type*, this can change the load the circuit from what is shown in the picture, to a perfect diode where no parasitic components are shown to a device at open circuit which would be used to simulate Transient Photo Voltage measurements. The right hand figure shows the configuration options of the time domain window. Again notice the *Output verbosity to disk* option as described in the previous section, you will see this again and again in OghmaNano.

4.2. TIME DOMAIN EDITOR

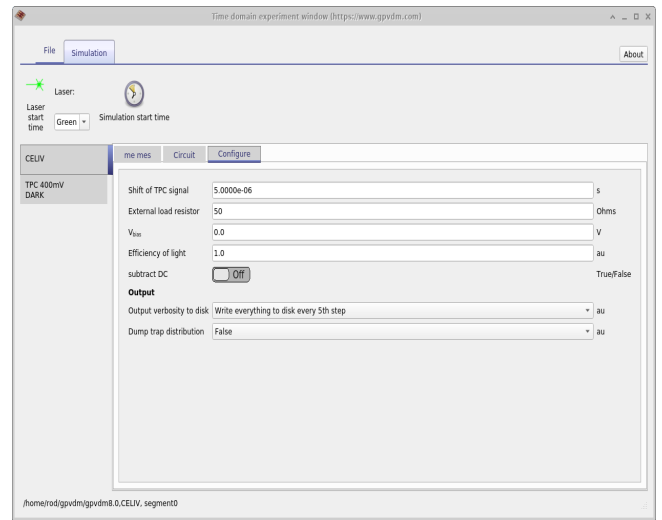
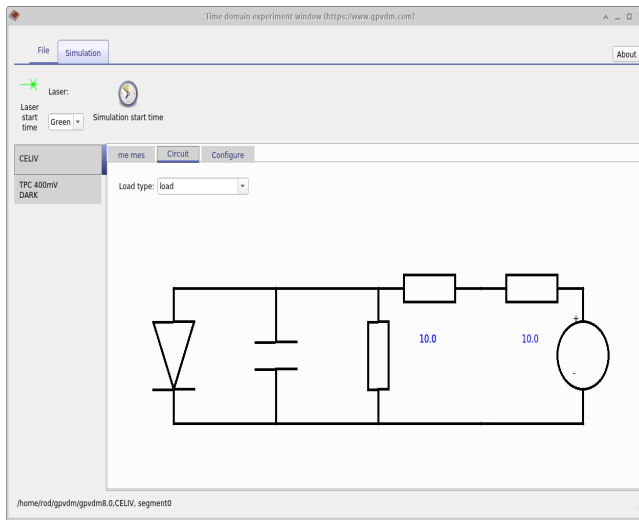


Figure 4.8: Configuring the time domain editor: Left the circuit diagram used by the time domain window and right simulation options.

4.3 Frequency domain editor

Related YouTube videos:



Simulating impedance spectroscopy (IS) in solar cells.

4.3.1 Overview

The frequency plugin allows you to simulate the frequency domain response of the device. Using this tool one can perform impedance spectroscopy, as well as optically excited measurements such as Intensity Modulated Photo Spectroscopy (IMPS), Intensity Modulated Voltage Spectroscopy (IMVS). The domain editor allows you to configure frequency domain simulations. This is shown below in Figures 4.9 and 4.10. On the left hand side is the frequency domain mesh editor this is used to define which frequencies will be simulated. Figure 4.10 shows the *circuit* tab of the frequency domain window, this sets the electrical configuration of the simulation. One can either simulate an ideal diode (this is the fastest type of simulation to perform), a diode with parasitic components or a diode in open circuit. An ideal diode would be used for IMPS simulations while the open circuit model would be used for IMVS simulations. Pick the circuit depending on what conditions you want to simulate. If you want examples of frequency domain simulation look in the new simulation window under Organic Solar cells, some of the PM6:Y6 devices have examples of frequency domain simulations already set up.

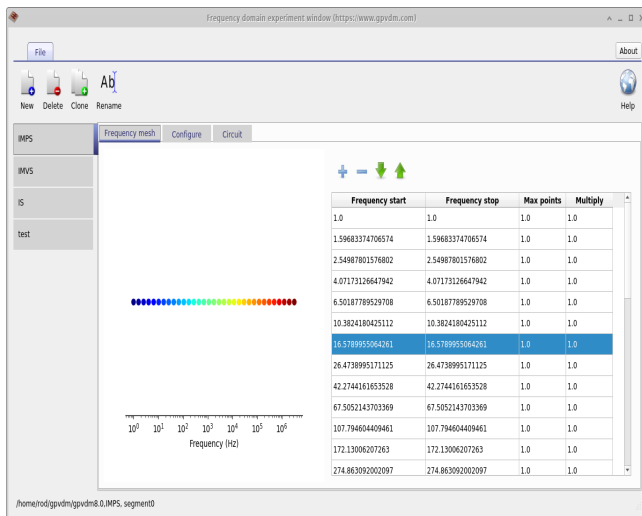


Figure 4.9: The frequency domain editor window

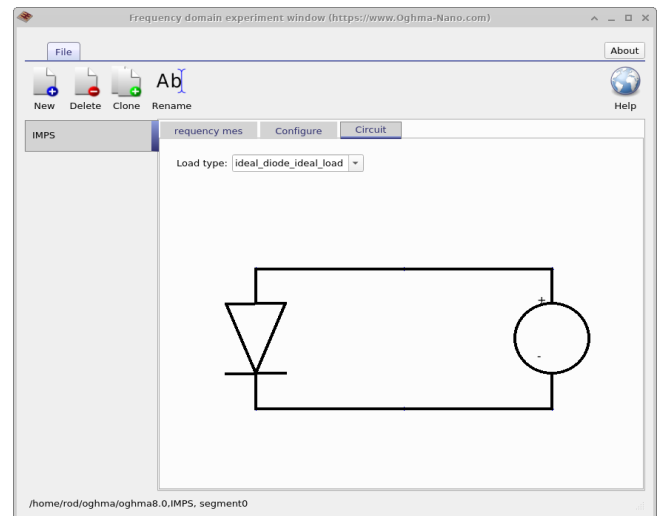


Figure 4.10: A circuit set up for frequency domain simulations.

Large signal or small signal

There are two ways to simulate frequency domain simulations in a device model, a large signal approach or a small signal approach. The small signal approach assumes the problem we are looking at varies linearly around a DC point, this may or may not be true depending on the conditions one is looking at. This method is however computationally fast. The second approach is to use a large signal approach and rather than simulating linear variation around a set point one simulates the time domain response of the device in full for each wavelength of interest. This method is cope better non-linear systems and one does not need to worry if one is in the large or small signal regime but is slower. OghmaNano uses the large signal approach.

4.3. FREQUENCY DOMAIN EDITOR

File name	Description
$V_{external}$	The external voltage applied to the cell
Simulation type	Leave this as Large signal.
Load resistor	External load resistor, this should be usually set to zero.
FX domain mesh points	The number of time steps used to simulate each cycle
Cycles to simulate	The number of complete periods of any given frequency that are simulated
Excite with	How the device is excited, either optically or electrically.
Measure	What is measured, current or voltage.
Modulation depth	How deep is the DC voltage/current modulated
Periods to fit	The number of frequency domain cycles that are fit to extract phase angle
Output verbosity to disk	How much data is dumped to disk (described in other sections)
Output verbosity to screen	How much data is shown on the creen (described in other sections)

Table 4.2: Files produced by the time domain simulation

4.3.2 Inputs

In Figure 4.11 the *Configure* tab of the frequency domain window can be seen. This decides exactly how the simulation will perform. These are described below in table 4.2

4.3.3 Outputs

File name	Description
real_imag.csv	Re(i(fx)) v.s. Im(i(fx))
fx_imag.csv	fx v.s. Im(i(fx))
fx_real.csv	fx v.s. Re(i(fx))
fx_abs.csv	fx v.s. $ i(fx) $
fx_phi.csv	fx v.s. $\angle i(fx)$
fx_C.csv	fx v.s. Capacitance
fx_R.csv	fx v.s. Resistance

Table 4.3: Files produced by the time domain simulation

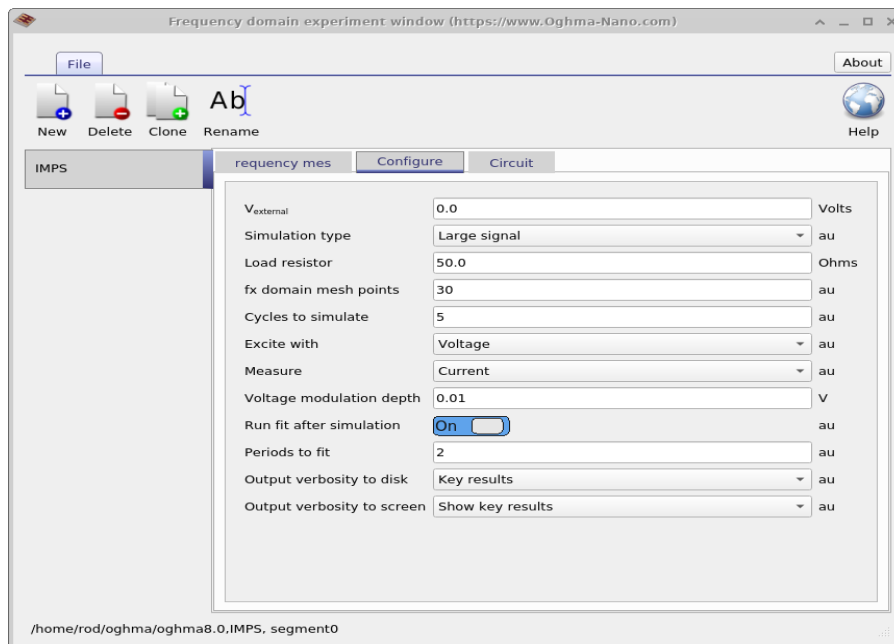


Figure 4.11: Configuring a frequency domain simulation

4.4 Suns-Voc editor

The Suns-Voc plugin can be used to calculate how open circuit voltage changes as a function of light intensity. This can be useful for understanding tail slope and disorder in devices. A picture of the suns-voc editor window can be seen below in figure 4.12. The window can be used to set the start and stop light intensity. The Suns-Voc applies the voltage to the contact that is labelled *Change*.

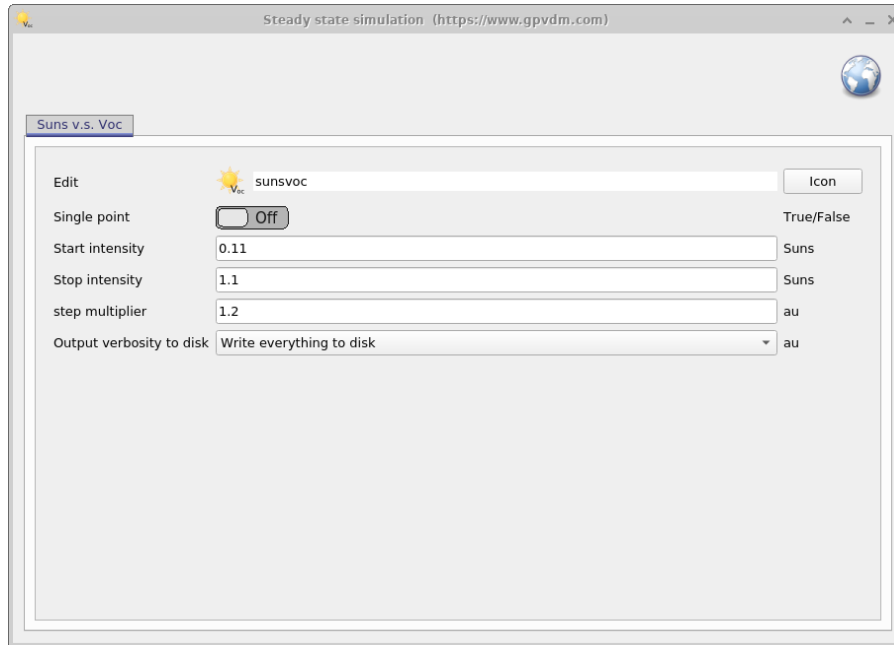


Figure 4.12: The suns-voc editor window

4.4.1 Outputs

File name	Description
suns_voc.csv	Suns v.s. Voc curve
suns_Q.csv	Suns v.s. Charge density
suns_mu.csv	Suns v.s. average charge carrier mobility
suns_tau.csv	Suns v.s. recombination constant tau
Q_Qtau.csv	Charge density v.s. recombination constant tau
Q_mu.csv	Charge density v.s. charge carrier mobility
Q_kbi.csv	Charge density v.s. recombination prefactor kbi
Q_trap_filling.csv	Charge density v.s. fraction of filled traps
V_mu.csv	Voc v.s. average charge carrier mobility

Table 4.4: Files produced by the Suns-Voc simulation

4.5 Suns-Jsc editor

The Jsc editor can be used to configure suns-Jsc simulations. It enables you to set the start light intensity, stop light intensity and how big the steps are. This is shown in figure 4.13.

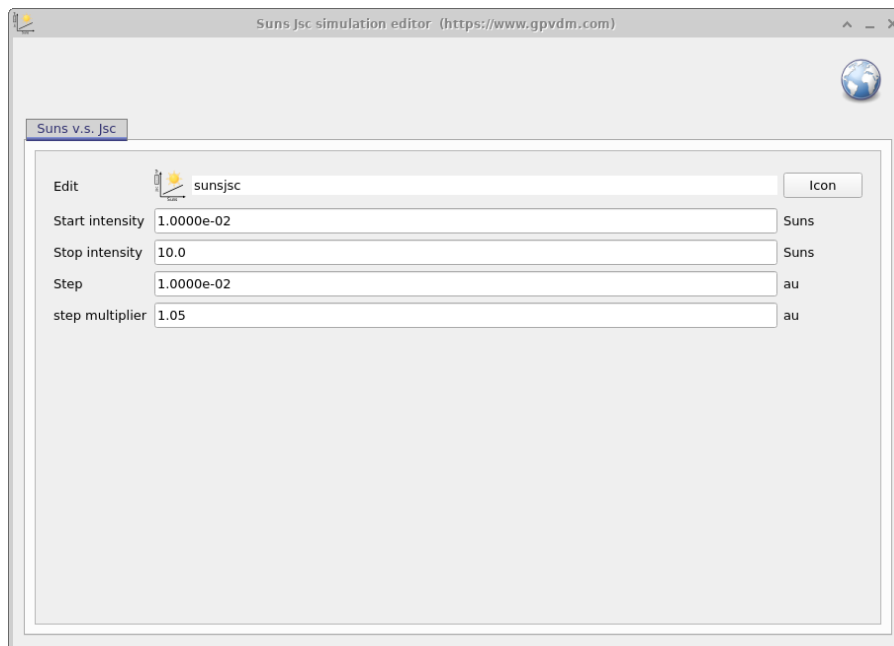


Figure 4.13: The JV curve editor window

4.5.1 Outputs

File name	Description
suns_jsc.csv	Suns v.s. Jsc curve
suns_mu.csv	Suns v.s. average charge carrier mobility

Table 4.5: Files produced by the Suns-Jsc simulation

4.6 Quantum efficiency editor

The quantum efficiency editor simulates both EQE and IQE. The configuration window can be used to set the voltage at which EQE and IQE are performed.

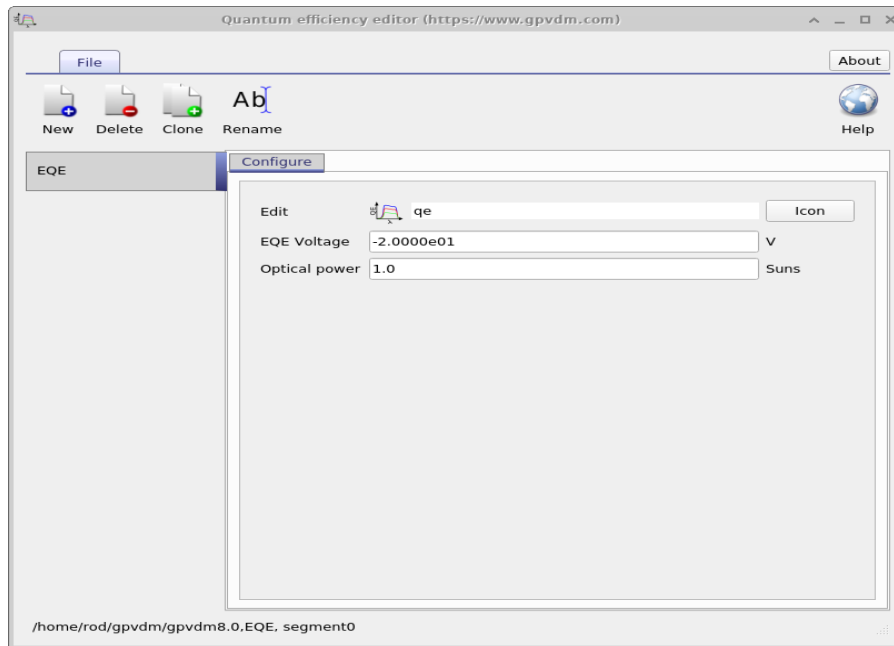


Figure 4.14: The quantum efficiency editor window

4.6.1 Outputs

File name	Description
eqe.csv	Wavelength v.s. EQE
E_eqe.csv	Photon energy v.s. EQE
E_eqe_norm.csv	Photon energy v.s. Normalized EQE
iqe.csv	Wavelength v.s. IQE
E_iqe.csv	Photon energy v.s. IQE
lam_Gn.csv	Wavelength v.s. Average charge carrier generation rate

Table 4.6: Files produced by the Suns-Jsc simulation

4.7 Scanning probe microscopy editor

When simulating a 3D structure such as a large area contact one often wants to map the resistance between an x,z point on the surface of the device and the charge extraction contact. This tool is used to apply voltages systematically over z,x regions to map out voltage or resistance profiles in space. This tool is usually used with either full 2/3D drift diffusion simulations or the 3D large area electrical circuit model. There is more about this tool in Section 11.

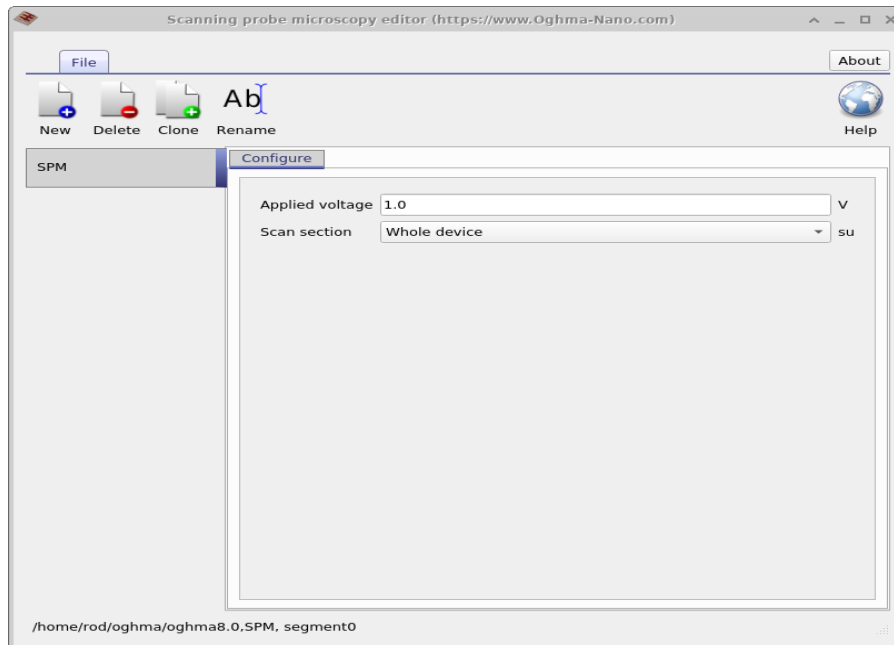


Figure 4.15: The scanning probe microscopy editor

4.8 Electrical equilibrium editor

Sometimes when studying a device, it is not necessary to simulate an entire JV curve. One may for example just for example be interested in the band structure at 0V in the dark. The *Electrical equilibrium* allows the user to setup simulations that only simulate the device at equilibrium (0V applied bias in the dark).

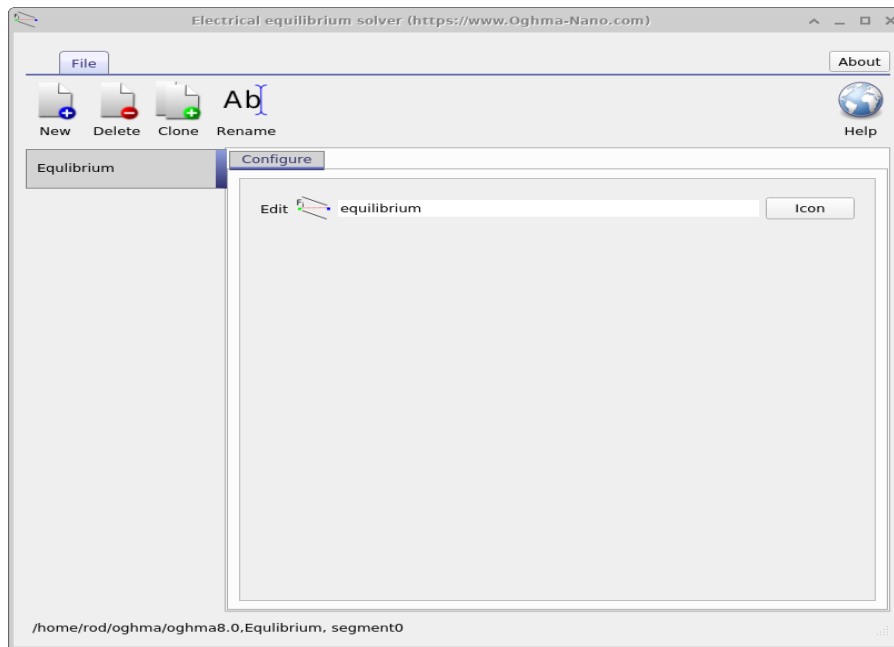


Figure 4.16: Electrical equilibrium editor

4.9 Steady state photoluminencense editor

This tool is used to generate photoluminencense spectra at a desired voltage, either short circuit or open circuit.

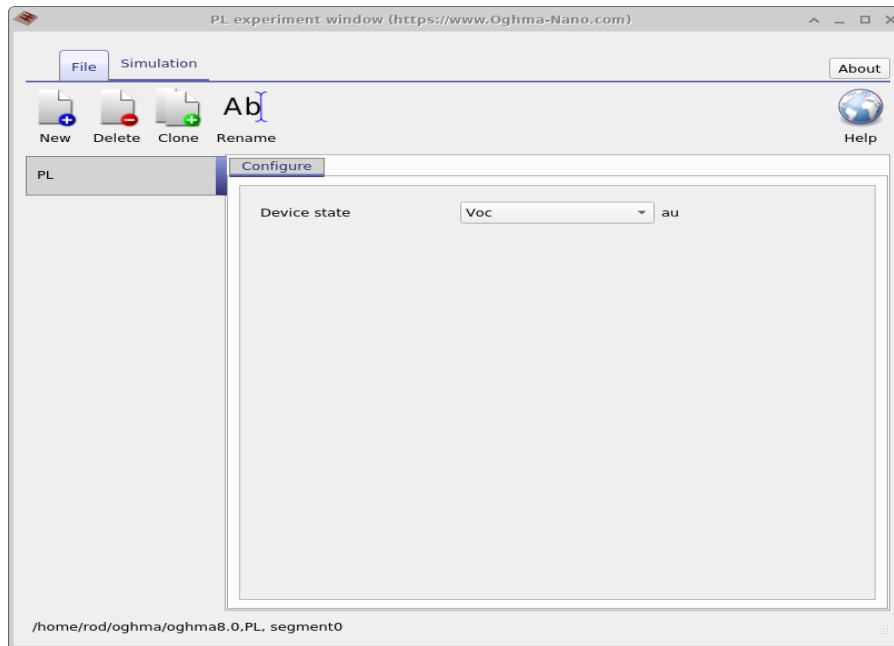


Figure 4.17: Steady state photoluminencense editor

4.10 Charge extraction editor

This is the charge extraction editor, it allows one to simulate charge extraction transients. A charge extraction experiment is performed to find out how much charge is in a disordered device. For this type of experiment one runs the device at a set voltage and light intensity say 1V @ 1Sun. Then one turns off the light and shorts the cell through a resistor and integrates the total current outputted by the cell to get the charge that was in the cell when it was operating. Typically the cell is shorted through the the 50 Ohm termination of an oscilloscope so that by measuring the voltage transient and applying $V=IR$ one can calculate the current and thus total charge.

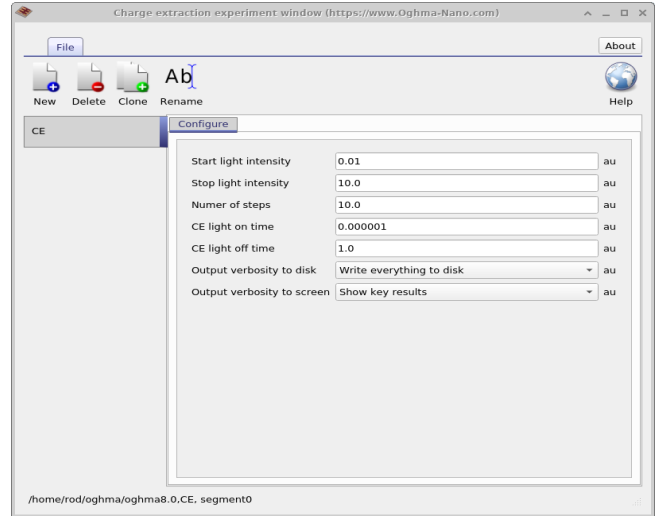


Figure 4.18: Charge extraction editor

This type of measurement is important in disordered devices when one wants to find out how much charge is in the trap states. It is important to note that the experiment does not extract all the charge in the device, it only extracts the difference between charge at operating conditions and 0V @ 0 Suns. The background charge due to injection from the contacts/doping is still left in the device. There is also error loss in the CE experiment due to recombination annihilating charge before it has left the device.

4.10.1 Outputs

File name	Description
<i>time_i.csv</i>	Time v.s. extraction current for a single CE experiment
<i>time_v.csv</i>	Time v.s. voltage for a single CE experiment
<i>suns_Q.ce.dat</i>	Suns v.s. extracted charge including effects of recombination
<i>v_np.dat</i>	Voltage c.s. extracted charge including the effects of recombination
<i>suns_np.dat</i>	Suns c.s. extracted charge including the effects of recombination
<i>v_np_ideal.dat</i>	Voltage v.s. extracted charge not including the effects of recombination
<i>suns_np_ideal.dat</i>	Suns v.s. extracted charge not including the effects of recombination

Table 4.7: Files produced by the charge extraction simulation

4.11 Capacitance voltage editor

Experimentally capacitance voltage (CV) measurements are a useful way to determine doping within a device. In OghmaNano CV measurements use a cut down version of frequency domain simulation tool described above.

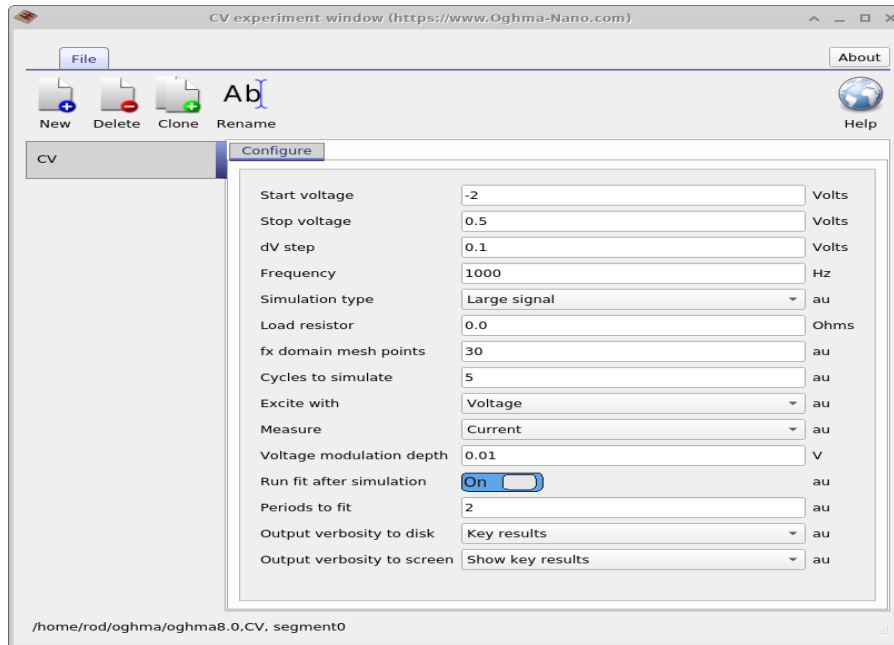


Figure 4.19: The capacitance voltage editor

4.11.1 Outputs

File name	Description
real_imag.dat	$\text{Re}(i(fx))$ v.s. $\text{Im}(i(fx))$
fx_real.dat	fx v.s. $\text{Re}(i(fx))$
fx_imag.dat	fx v.s. $\text{Im}(i(fx))$
cv.dat	fx v.s. Capacitance
cv2.dat	fx v.s. $1/\text{Capacitance}^2$

Table 4.8: Files produced by the CV simulation

Chapter 5

2D Simulations - OFETs



Tutorial on OFET simulation.

OghmaNano contains a 2D electrical solver that can be used for simulating OFETs and other 2D structures. To perform 2D simulations use the default OFET simulation in OghmaNano as a starting point. You can do this by double clicking on *OFET simulation* in the new simulation window (see figure ??).

Note: The 2D electrical solver is a separate plug in to the 1D solver, if you select the default OFET simulation OghmaNano as a starting point for your own 2D simulations OghmaNano will be all set up to do 2D electrical simulations. If you try to convert a 1D simulation such as a solar cell to a 2D simulation (not recommended) please read section 8.8 on how to select the correct solver.

To make a new OFET simulation, click on the new simulation button. In the new simulation window and select the OFET simulations (see figure 5.1). Double clicking on this will bring up the OFET sub menu, where other types of OFETs are also stored. There is one example with a top contacts, one with side contacts and one which is at low temperature. For this chapter we will be looking at the (standard) top contact OFET (Figure ??), double click on this and save the new simulation to disk.

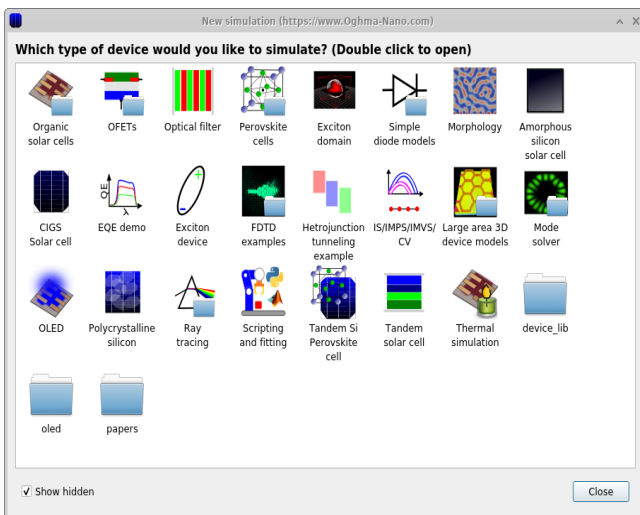


Figure 5.1: Select the OFET submenu to main a new OFET simulation

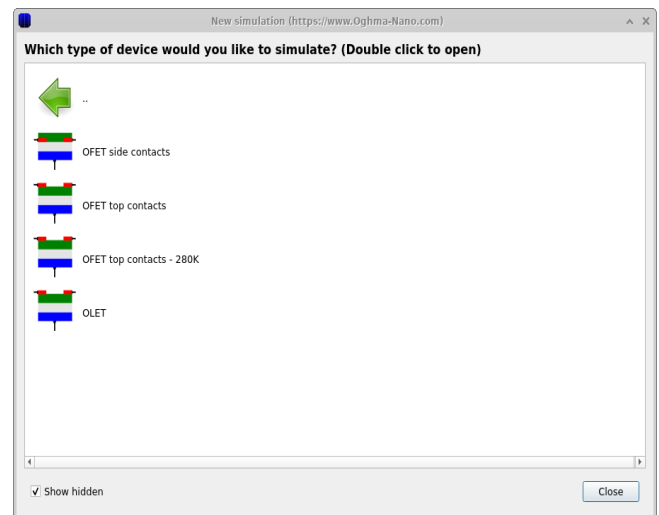


Figure 5.2: Select the OFET top contact for this example.

5.0.1 The anatomy of a 2D simulation

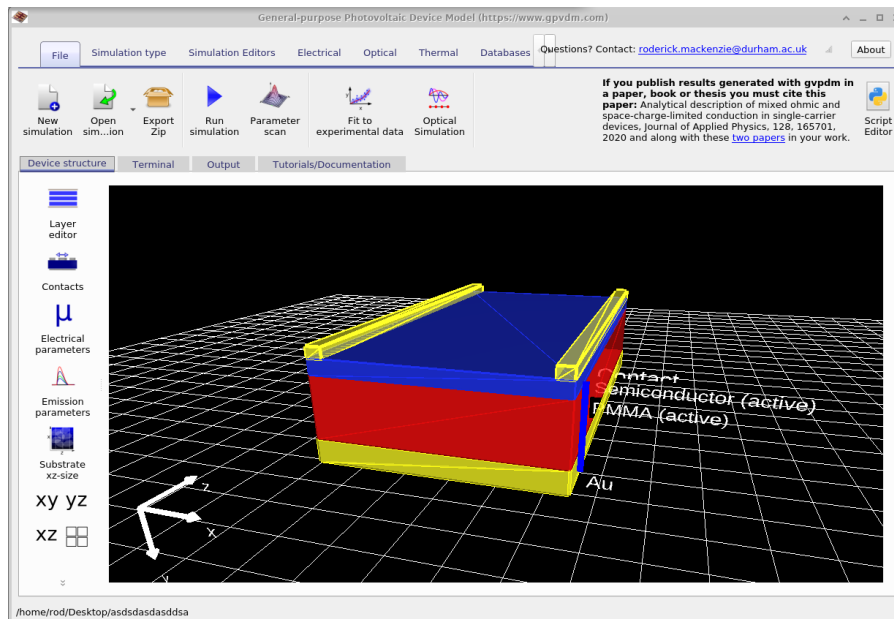


Figure 5.3: The default ofet simulation.

The OFET structure shown in Figure 5.3 has three contacts, a *gate*, *source* and a *drain*. The source and drain are shown on the top of the simulation as gold bars, a semiconductor layer is shown in blue and an insulating later shown in red. The *gate* contact is visible at the bottom of the structure. This layered structure is defined in layer editor, see figure 5.4. The layer editor has been described in detail in section 3.1.3. It can be seen that the top and bottom layers have been set to *contact* and the insulator (PMMA) and semiconducting layer have been set to *active*. This means that the electrical model will only consider the semiconductor and insulator layers and the contacts will be used as boundary conditions. As this structure is not emitting light the *Optical material* column has no impact on the simulation results so it has been arbitrary materials.

Layer name	Thickness (m)	Optical material	Layer type	Solve optical problem	Solve thermal problem	ID
Contact	7e-08	generic/air	contact	Yes	Yes	...
Semiconduc...	1e-07	generic/n/2.0	active layer	Yes	Yes	...
PMMA	4e-07	generic/n/2.0	active layer	Yes	Yes	...
Au	1.5e-07	generic/n/2.0	other	Yes	Yes	...

Figure 5.4: The layers of an OFET device

The contacts are defined in the contact editor shown in Figure 5.5. The contact editor has been described in detail in section 3.1.8, however because this is a 2D simulation another two extra columns have appeared. They are *start* and *width*. These define the start position of the contact on the x-axis and width which describes the width of the contact on the x-axis. The *source* starts at 0 m and extends to $5\mu\text{m}$, the *drain* starts at $75\mu\text{m}$ and extends to $5\mu\text{m}$, while the gate starts at 0 m and extends to cover the entire width of the device which is $80\mu\text{m}$. If you are unsure which is the x-axis, the origin marker is visible at the bottom of figure 5.3. Notice also that under the column *Applied Voltage*, the *source* is marked *Ground* this means that 0V will be applied to the ground, the *gate* is marked *change* meaning that our voltage ramp as defined in the JV editor will be applied to this contact, and the *drain* is marked *constant bias* with a voltage of 15V, this means that a constant voltage of 15V will be applied to this contact. And thus we are scanning the gate contact while applying a constant voltage between the source and the drain.

Name	Top/Bottom	Applied voltage	Start (m)	Width (m)	Charge density/ Fermi-offset	Majority carrier	Physical model	
Source	top	Ground	Gnd	0.0	5e-06	1.38e18 m ³ / 0.45 eV	Electron	Ohmic
Gate	bottom	Change	Vsig	0.0	8e-05	9e20 m ³ / 0.28 eV	Hole	Ohmic
Drain	top	Constant bias	15.0	7.5e-05	5e-06	1.38e18 m ³ / 0.45 eV	Electron	Ohmic

Figure 5.5: Editing the contacts on a 2D device.

5.0.2 Electrical parameters

Disabling drift diffusion in the insulator layer

The electrical parameters for both the semiconductor and the insulator can be seen in Figure 5.6, these can be accessed through the *Electrical parameter* editor. The *Electrical parameter* editor is described in detail in section 3.1.9. The left image shows the parameters for the semiconducting layer while the right figure shows the parameters for PMMA. If you look in the top left of both windows you will see a button called *Enable Drift Diff.* which stands for *Enable Drift Diffusion*. When this is depressed the drift diffusion equations will be solved within the layer which take into account charge movement. When this is not depressed only Poisson's equation will be solved in the layer and the movement of charge ignored. If you notice this button is depressed in the Semiconductor layer and not depressed in the PMMA insulator. This means that the drift-diffusion equations will be solved in the semiconductor and not in the PMMA. The reason for doing this is that charge does not conduct in the PMMA so there is no point in solving the drift-diffusion equations in that layer. Another approach would be to solve the drift-diffusion equations in both layers and just set the mobility in the PMMA to be very low but this will result in slower computational times and is less numerically stable, there is more on this approach below.

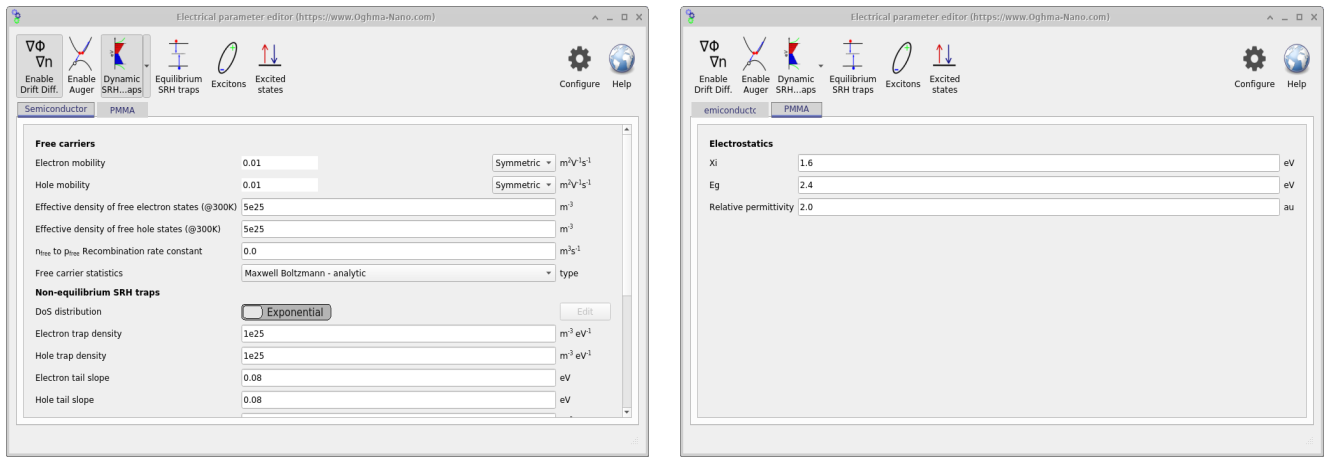


Figure 5.6: Electrical parameters for both the semiconductor (left) and the insulator (right)

5.0.3 Running a 2D simulation

2D simulations are run in the same way as 1D simulations, simply click on the play button, see figure 5.7.

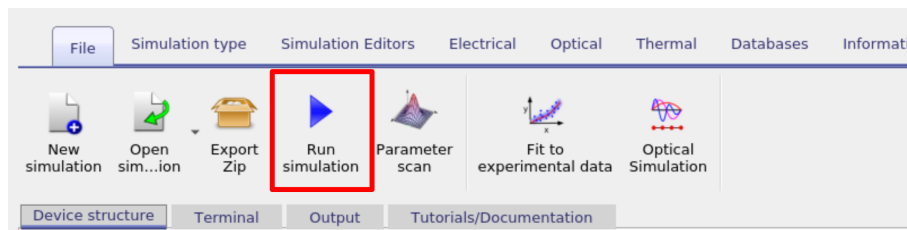


Figure 5.7: Running an OFET simulation

The simulation will take longer than its 1D counterparts simply because there will be more equations to solve. If you have set a contact at a high starting voltage the solver will initially ramp the contact voltage in a stepwise way until the desired voltage is achieved before the desired voltage sweep is applied to the *active contact*. After the simulation has run the following files will be produced showing the current density from each contact.

File name	Description
<i>contact_iv0.dat</i>	Current voltage current curve for contact 0
<i>contact_iv1.dat</i>	Current voltage current curve for contact 1
<i>contact_iv2.dat</i>	Current voltage current curve for contact 2
<i>contact_jv0.dat</i>	Current density voltage current curve for contact 0
<i>contact_jv1.dat</i>	Current density voltage current curve for contact 1
<i>contact_jv2.dat</i>	Current density voltage current curve for contact 2
snapshots	Simulation snapshots

Table 5.1: Files produced by the steady state OFET simulation

Contacts in OghmaNano are labelled from 0 to N in the order they are defined in the contact editor (see Figure 5.5), so in this case Contact 0 will be the source, contact 1 will be the gate and contact 2 will be the drain. You can see the result of running the simulation in Figure 5.8.

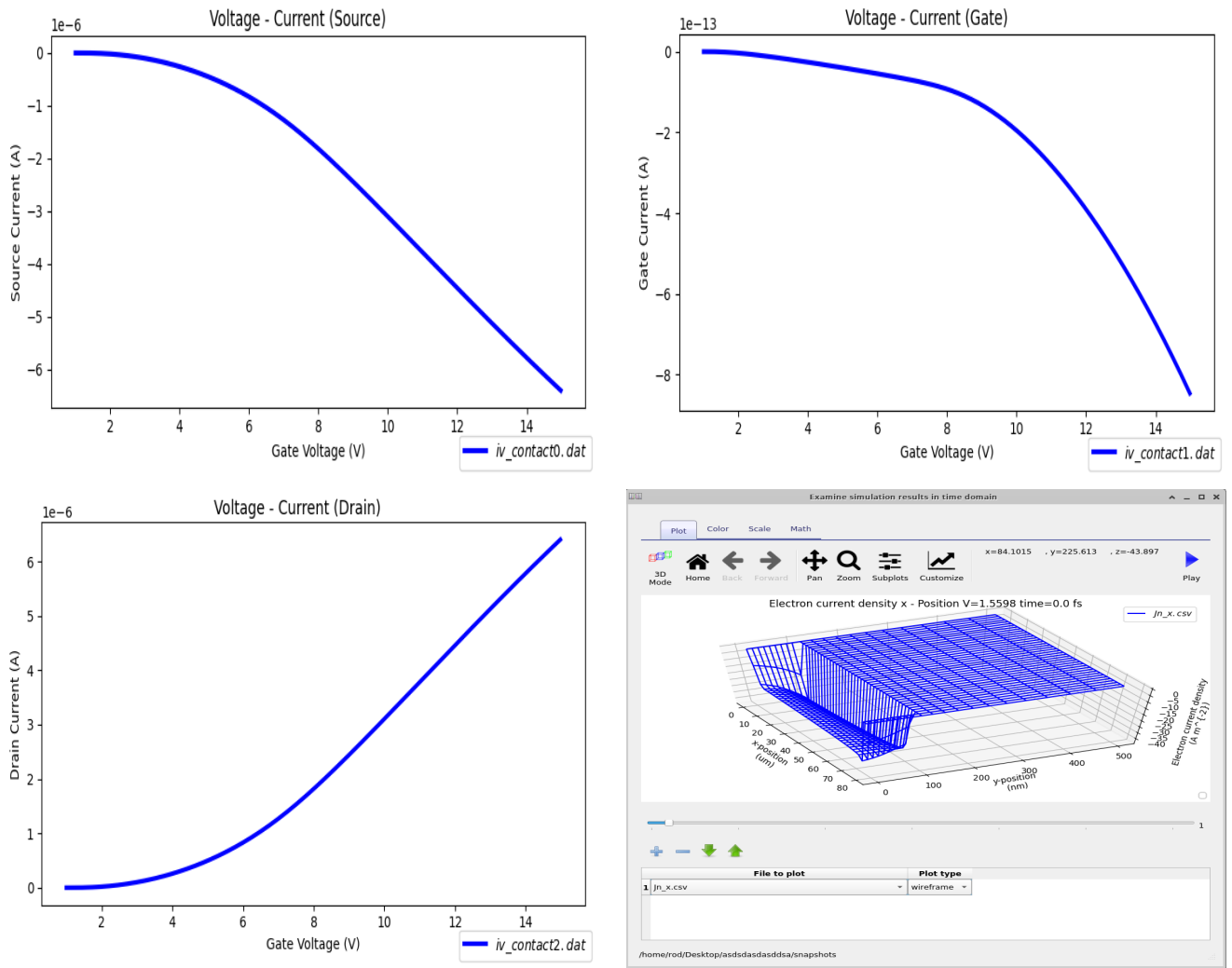


Figure 5.8: Results from a 2D OFET simulation the JV curves for each contact are shown along with a view of the the electron current density in the x direction (bottom right).

5.0.4 Meshing in 2D

Computational speed, traps and meshing

You will notice that in this example the SRH trapping/escape equations are solved in the semiconductor layer, you can see this as the SRH trap button is depressed. Trapping is often needed to reproduce experimental results. If you scroll down the parameters list in the Semiconductor layer you will see that it has 8 trap states. However, it is worth taking a moment to consider the computational load of introducing trap states. If our 2D device has N_x mesh points in the x direction and N_y mesh points in the y direction then and we are solving Poisson's equation, the electron drift diffusion equation and the hole drift diffusion equation then we will be solving $3 * N_x * N_y$ equations in total. If we now introduce 8 trap states for electrons and 8 trap states for holes we will then be solving $3 * N_x * N_y + 8 * 2 * N_x * N_y$ equations. So if you want a speedy simulation or are just trying something out it is worth trying to reduce the number of mesh points, and also reduce the number of trap states and/or turn traps off in the first instance.

Adjusting the mesh

The electrical mesh editor can be accessed through the electrical ribbon in the main window. The mesh editor is shown in Figure 5.9, here the x and y mesh can be adjusted. The number

of mesh points directly affects the speed of the computation, as a general rule try to minimize the number of mesh points you use. I would recommend defining one electrical mesh to cover the Semiconductor layer and one to cover the insulator layer.

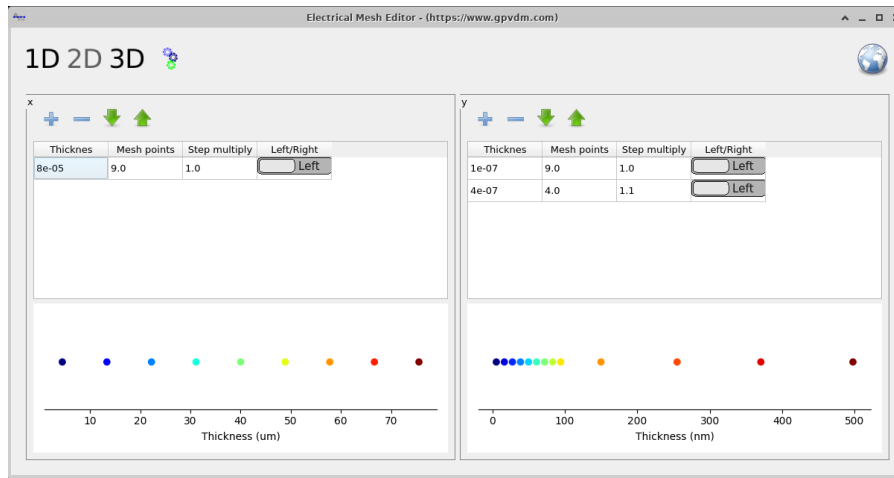


Figure 5.9: Meshing

5.0.5 Solving the drift diffusion equations over the entire device

Sometimes you may want to solve the drift diffusion equations over the entire device this could be because you have a poor insulator on the gate contact, it is very uncommon to want to do this but if you want to follow the steps below. Using doing this is not recommended.

- Mobility: Set the mobility of the insulator to a value of $1 \times 10^{-12} - 1 \times 10^{-15} m^2 / (Vs)$ to limit current flow into the region. However, the value should not be set too low (see section 8.8.1) or the solver may become numerically unstable.
- Effective density of states: Keep these the same for both layers, just to keep things simple.
- Number of trap states: This must be the same in both layers, the density of the states and the Urbach energy can change though.
- Eg and Xi: Although it is tempting to simply enter the experimental values for Xi and Eg for both the insulator and the semiconductor, one has to be careful in doing this as some insulators (SiO_2) have very big band gaps which mean the number of carriers get very small and make the simulation unstable (read section 8.8.1 for an explanation). If you want to simulate a jump in the band gap into an insulator, my is to make the jump significantly bigger than $3/2kT = 25meV$ which is the average kinetic energy of a charge carrier. If the gap is between $0.5 - 1.0V$ charge carriers will have problems penetrating the barrier and there is no need to simulate bigger steps.

Chapter 6

2D simulation of bulk-heterojunctions



Simulating 2D BHJ structures in OghmaNano

To be written but there is an example simulation in the new simulation window.

Chapter 7

Meshing

7.1 Meshing

The thermal optical and electrical ribbons are shown below in Figure 7.1, it can be seen that in each of these ribbons is a a mesh button, where the thermal, optical and electrical meshes can be defined. OghmaNano in principle describes the physical problems

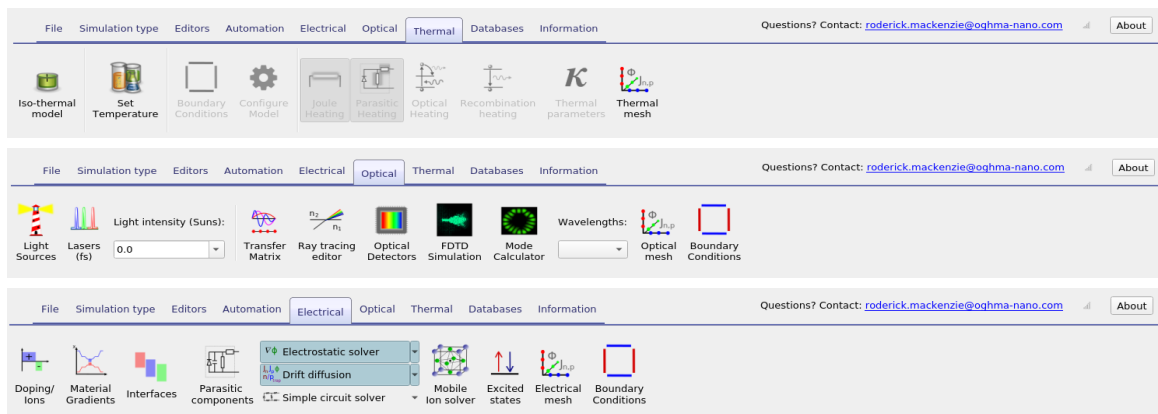


Figure 7.1: The thermal, electrical and optical ribbons.

7.2 Editing the electrical mesh/layers

The device structure is split up into layers of different materials. These can be configured in layer editor which is discussed in section 3.1.3. Some of these layers will have the layer type 'active'. An 'active' layer is a layer over which the electrical model will be applied. The electrical model needs a finite difference mesh to be setup for it to work. Usually, this will be taken care of automatically by OghmaNano. However, some users will want fine control over the mesh. This section describes how to do that. The electrical mesh editor is depicted in figure 7.3.

The buttons marked 1D, 2D and 3D at the top of the window can be used to toggle the simulation between 1D, 2D and 3D modes. (Note, if you want to do 2D or 3D simulations you are best off using a default 2D simulation, such as the OFET simulation. This is because to do 2D/3D simulations, a special newton solver configuration will be needed.) The table on the left hand side is used to configure the mesh. The sum of the mesh layer thicknesses must exactly match that of the sum of the active layers. If this is not the case, the model will give an error message. The columns thickness and mesh points, determine the thickness of the mesh layer and the number of points on the mesh layer, if there is a uniform spacing between mesh points.

7.2. EDITING THE ELECTRICAL MESH/LAYERS

The column, 'step multiply' by how much to grow each step. In this example, the mesh spacing is increased by a factor of 0.1 each step. The toggle button left/right, defines on which side the mesh layer is generated. In this example there are two mesh layers, one starting on the left and one starting on the right. The resulting mesh is plotted in the graph at the bottom of the window. It can be seen that a non-linear mesh has been generated.

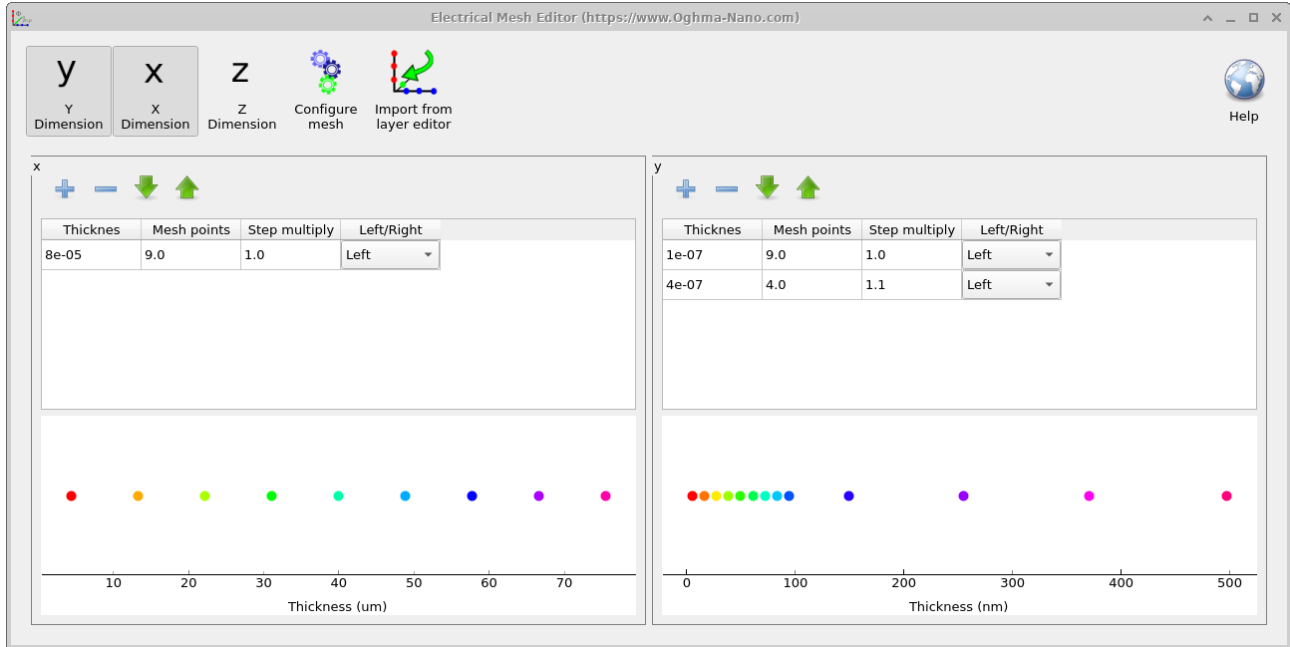


Figure 7.2: The electrical mesh editor

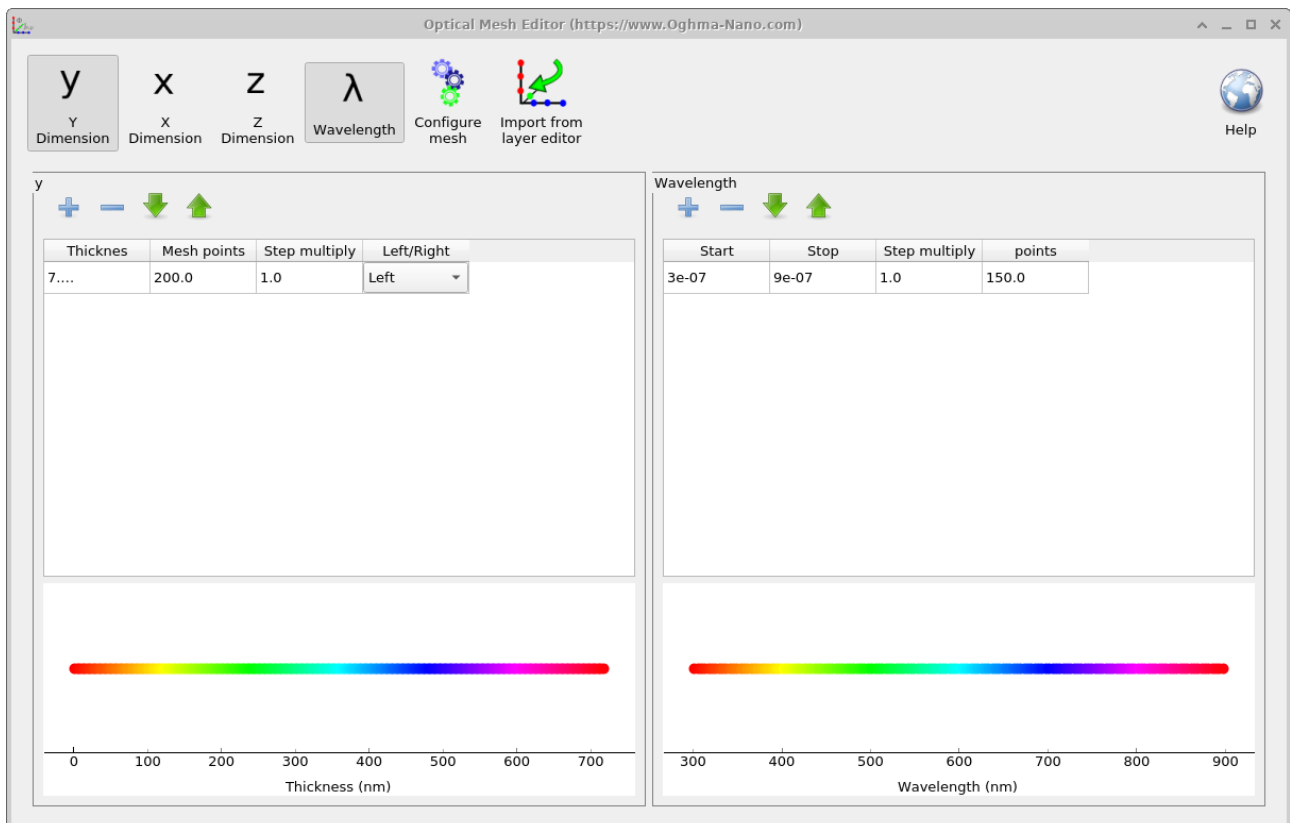


Figure 7.3: The electrical mesh editor

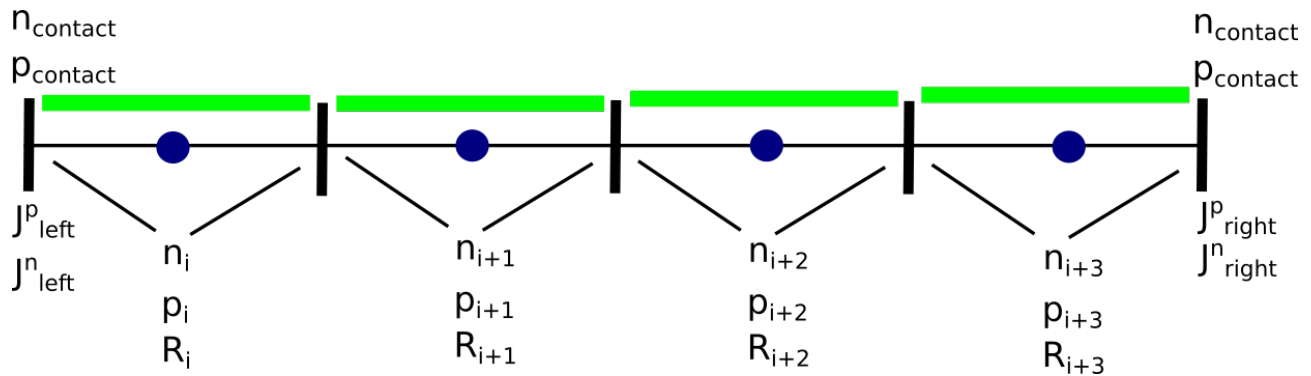


Figure 7.4: A 1D diagram of the mesh

7.3 Should I be simulating in 1D, 2D or 3D?

When deciding if you should perform 1D, 2D or 3D, simulations, consider the dimensionality of your problem. For example if you consider a solar cell, it is only a few microns thick, and there is rapid variation in the structure, charge densities, mobilities, and doping as a function of depth (y). However, the structure will not vary very in the lateral (xz) plane. Therefore, in general to capture all interesting effects present within a solar cell one only needs a 1D model. If one now considers OFETs, there is both vertical and lateral current flow, therefore one can not get away with a 1D model any more, as one must simulate both vertical current flow, and current between the source and the drain, thus one needs a 2D simulation. As the number of dimensions increases, computation speed will decrease, therefore my general advice is to use the minimum number of dimensions possible to solve your problem.

In short try to make your simulation as simple as possible as it will save you time and effort. Generally the following geometries could be used for various types of devices:

Device type	Number of dimensions
<i>Solarcells</i>	1D
<i>Optical filter</i>	1D
<i>OFET</i>	2D

Table 7.1: How many dimensions should I use to simulate my device.

Chapter 8

Theory of drift diffusion modelling

8.1 Outline

OghmaNano's electrical model is a 1D/2D drift-diffusion model (like many others) however the special thing about OghmaNano which makes it very good for disordered materials (Think organics, perovskites and a-Si) is that it goes to the trouble of explicitly solving the Shockley-Read-Hall equations as a function of energy and position space. This enables one to model effects such as mobility/recombination rates changing as a function of carrier population and enables one to correctly model transients as one does not have to assume all the carriers in the trap states have reached equilibrium. Things such as ToF transients, CELIV transients etc.. can be modelled with ease. Of course can be used for more ordered materials as well, you then just need to turn the traps off.

8.2 Electrostatic potential

The conduction band/valance band (or LUMO/HOMO in organic semiconductor speak) are defined as

$$E_{LUMO} = -\chi - q\phi \quad (8.1)$$

$$E_{HOMO} = -\chi - E_g - q\phi \quad (8.2)$$

To obtain the internal potential distribution within the device Poisson's equation is solved,

$$\nabla \cdot \epsilon_0 \epsilon_r \nabla = q(n_f + n_t - p_f - p_t - N_{ad} + -N_{ion} + a), \quad (8.3)$$

where n_f , n_t are the carrier densities of free and trapped electrons; p_f and p_t are the carrier densities of the free and trapped holes; and N_{ad} is the doping density. N_{ion} is the background density of perovskite ions and a is the density of mobile ions.

8.3 Free charge carrier statistics

For free carriers the model can either use Maxwell-Boltzmann statistics i.e.

$$n_l = N_c \exp\left(\frac{F_n - E_c}{kT}\right) \quad (8.4)$$

$$p_l = N_v \exp\left(\frac{E_v - F_p}{kT}\right) \quad (8.5)$$

or full Fermi-dirac statistics i.e.

$$n_{free}(E_f, T) = \int_{E_{min}}^{\infty} \rho(E) f(E, E_f, T) dE \quad (8.6)$$

$$p_{free}(E_f, T) = \int_{E_{min}}^{\infty} \rho(E) f(E, E_f, T) dE \quad (8.7)$$

where

$$f(E) = \frac{1}{1 + e^{E - E_f / kT}} \quad (8.8)$$

When using FD statistics free carriers are assumed to move in a parabolic band:

$$\rho(E)_{3D} = \frac{\sqrt{E}}{4\pi^2} \left(\frac{2m^*}{\hbar^2}\right)^{3/2} \quad (8.9)$$

The average energy of the carriers is defined as

$$\bar{W}(E_f, T) = \frac{\int_{E_{min}}^{\infty} E \rho(E) f(E, E_f, T) dE}{\int_{E_{min}}^{\infty} \rho(E) f(E, E_f, T) dE} \quad (8.10)$$

8.4 Carrier trapping and Shockley-Read-Hall recombination

The model provides two methods to account for carrier trapping and recombination via trap states. The first by equation 8.11, this assumes that the trapped carrier distribution has reached equilibrium. It also assumes there are relatively few trapped charge carriers compared the the number of free carriers, and thus the trapped charges do not significantly change the electrostatic potential. These assumptions are valid when the material is very ordered (i.e. GaAs) or at a push in steady state for some moderately disordered material systems. However if you wish to simulate transient or frequency domain experiments, then you can no longer use 8.11. Instead, one must use a non-equilibrium SRH approach which does not assume trapped carriers have reached equilibrium. Unlike many other models, OghmaNano has such a non-equilibrium SRH model built in this is described in section 8.4.2. In fact, it is turned on by default so when using OghmaNano you have to go out of your way to turn on equation 8.11.

To understand the importance of such a dynamic solver, consider the following example: You are performing a transient photocurrent experiment (TPC). You photo-excite your device with a laser, carriers very quickly become trapped during the first 1-2 μ s after photoexcitation, as time passes, the carriers gradually de-trap from deeper and deeper trap states and produce the long photocurrent transient [10]. These transients can often extend out to over 1 second after photo-excitation. Current at the start of the transient originates from shallow traps while current at the end of the transient originates from carriers from very deep trap levels. To simulate this one has to be able to account for the gradual emptying of trap states firstly starting at the shallow traps, then progressing to deeper and deeper trap states. Were one to assume all trap states were in equilibrium one would not be able to simulate this process.

So in summary, although many others have used 8.11 to model disordered devices in time DON'T you results won't make sense. If you want to simulate anything but steady state in an

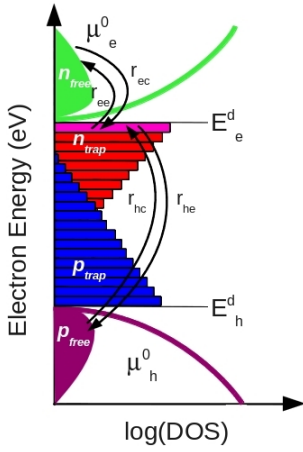


Figure 8.1: Trap filling in both energy and position space as the solar cell is taken from a negative bias Carrier trapping, de-trapping, and recombination

ordered device turn ON the non-equilibrium solver.

8.4.1 Equilibrium Shockley-Read-Hall recombination

For some very ordered material systems where there are not many trap states it is enough to describe SRH trap states using the equation:

$$R^{SRH} = \frac{np - n_0 * p_0}{\tau_p(n + n_1) + \tau_n(p + p_1)} \quad (8.11)$$

where R_{SRH} is the rate of SRH recombination, n, p are the density of free charge carriers n_0, p_0 , are the equilibrium density of charge carriers, $\tau_{n,p}$ are the SRH life times and n_1 and p_1 are the trapped electron and hole densities when the Fermi-level matches the trap state energy. This can be turned on in the electrical parameter editor.

8.4.2 Non-equilibrium carrier trapping and recombination using Shockley-Read-Hall trap states

To describe charge becoming trapping into trap states and recombination associated with those states the model uses Shockley-Read-Hall (SRH) theory. A 0D depiction of this SRH recombination and trapping is shown in figure 8.1, the free electron and hole carrier distributions are labeled as n free and p free respectively. The trapped carrier populations are denoted with n trap and p trap, they are depicted with filled red and blue boxes. SRH theory describes the rates at which electrons and holes become captured and escape from the carrier traps. If one considers a single electron trap, the change in population of this trap can be described by four carrier capture and escape rates as depicted in figure 8.1. The rate r_{ec} describes the rate at which electrons become captured into the electron trap, r_{ee} is the rate which electrons can escape from the trap back to the free electron population, r_{hc} is the rate at which free holes get trapped and r_{he} is the rate at which holes escape back to the free hole population. Recombination is described by holes becoming captured into electron space slice through our 1D traps. Analogous processes are also defined for the hole traps.

For each trap level the carrier balance 8.12 is solved, giving each trap level an independent quasi-Fermi level. Each point in position space can be allocated between 10 and 160 independent trap states. The rates of each process r_{ec} , r_{ee} , r_{hc} , and r_{he} are give in table 8.1.

$$\frac{\delta n_t}{\partial t} = r_{ec} - r_{ee} - r_{hc} + r_{he} \quad (8.12)$$

Mechanism	Symbol	Description
Electron capture rate	r_{ec}	$nv_{th}\sigma_n N_t(1-f)$
Electron escape rate	r_{ee}	$e_n N_t f$
Hole capture rate	r_{hc}	$pv_{th}\sigma_p N_t f$
Hole escape rate	r_{he}	$e_p N_t(1-f)$

Table 8.1: Shockley-Read-Hall trap capture and emission rates, where f is the fermi-Dirac occupation function and N_t is the trap density of a single carrier trap.

The escape probabilities are given by:

$$e_n = v_{th}\sigma_n N_c \exp\left(\frac{E_t - E_c}{kT}\right) \quad (8.13)$$

and

$$e_p = v_{th}\sigma_p N_v \exp\left(\frac{E_v - E_t}{kT}\right) \quad (8.14)$$

where $\sigma_{n,p}$ are the trap cross sections, v_{th} is the thermal emission velocity of the carriers, and $N_{c,v}$ are the effective density of states for free electrons or holes. The distribution of trapped states (DoS) is defined between the mobility edges as

$$\rho^{e/h}(E) = N^{e/h} \exp(E/E_u^{e/h}) \quad (8.15)$$

where, $N_{e/h}$ is the density of trap states at the LUMO or HOMO band edge in states/eV and where $E_U^{e/h}$ is slope energy of the density of states.

The value of N_t for any given trap level is calculated by averaging the DoS function over the energy (ΔE) which a trap occupies:

$$N_t(E) = \frac{\int_{E-\Delta E/2}^{E+\Delta E/2} \rho^e E dE}{\Delta E} \quad (8.16)$$

The occupation function is given by the equation,

$$f(E_t, F_t) = \frac{1}{e^{\frac{E_t - F_t}{kT}} + 1} \quad (8.17)$$

Where, E_t is the trap level, and F_t is the Fermi-Level of the trap. The carrier escape rates for electrons and holes are given by

8.4.3 Free-to-free carrier recombination

A free-carrier-to-free-carrier recombination (bi-molecular) pathway is also included. However, most organic solar cells have a great deal of trap states and an ideality factor greater than 1.0 suggesting that free-to-free recombination is not the dominant mechanism. Free-to-free recombination is described using equation 8.18

$$R_{free} = k_r(n_f p_f - n_0 p_0) \quad (8.18)$$

8.4.4 Auger recombination

Auger recombination is as

8.5. CHARGE CARRIER TRANSPORT

$$R^{AU} = (C_n^{AU} n + C_p^{AU} p)(np - n_0 p_0) \quad (8.19)$$

where C_n^{AU} and C_p^{AU} are the Auger coefficient of electrons and holes in $m^6 s^{-1}$. This can be set in the electrical paramter editor.

8.5 Charge carrier transport

To describe charge carrier transport, the bi-polar drift-diffusion equations are solved in position space for electrons,

$$\mathbf{J}_n = q\mu_e n_f \nabla E_c + qD_n \nabla n_f, \quad (8.20)$$

and holes,

$$\mathbf{J}_p = q\mu_h p_f \nabla E_v - qD_p \nabla p_f. \quad (8.21)$$

Conservation of charge carriers is forced by solving the charge carrier continuity equations for both electrons,

$$\nabla \mathbf{J}_n = q(R - G + \frac{\partial n}{\partial t}), \quad (8.22)$$

and holes

$$\nabla \mathbf{J}_p = -q(R - G + \frac{\partial p}{\partial t}). \quad (8.23)$$

where R and G are the net recombination and generation rates per unit volume respectively.

8.6 Perovskite mobile ion solver

The mobile ion solver is implemented after the work of Calado [11]

$$\mathbf{J}_a = q\mu_a a_f \nabla E_v - qD_a \nabla a_f. \quad (8.24)$$

$$\nabla \mathbf{J}_a = -q \frac{\partial a}{\partial t}. \quad (8.25)$$

8.7 Semiconductor interfaces

8.7.1 Tunnelling through heterojunctions

Tunnelling of holes through hetrojunction interfaces are is give by

$$\mathbf{J}_p = qT_h((p_1 - p_1^{eq}) - (p_0 - p_0^{eq})), \quad (8.26)$$

and for electrons

$$\mathbf{J}_n = -qT_e((n_1 - n_1^{eq}) - (n_0 - n_0^{eq})). \quad (8.27)$$

Where T_h and T_e represent the rate constants of the tunnelling. This can be configured in the interfaces editor.

8.7.2 Doping on the interface

Using the interface editor, layers of doping measuring one mesh point thick can be added to either side of the interface. This is useful for OFET simulations where interface charge is

important to the turn on voltage.

8.8 Configuring the electrical solver

Behind OghmaNano are a series of non-linear solvers that solve the electrical equations in a highly efficient way. These can be configured by going to the electrical tab. There you will see the Drift diffusion button, to the left of that is an arrow. If you click on this it will bring up a window which allows you to configure the "Newton solver". The options are described below.

Related YouTube videos:



How to optimize simulations in OghmaNano so they run faster

- Max Electrical iterations (first step): The maximum number of steps the solver can after it's cold started onto a new problem. This is usually at 0V in the dark. The solver usually takes more steps on it's first go.
- Electrical clamp (first step): This is a number by which the maximum newton step is clamped to. 0.1 will make the solver very stable but very slow, 4.0 will make the solver very fast but unstable. A recommended value of 1.0 is suggested for normal problems. If you are solving for high doping or other unusual conditions it can be worth reducing the step. Likewise if you want the solver to be fast and you know the problem is easy set the value to 2.0 or higher. For the first step, I would consider setting this value to be slightly lower than for the subsequent steps.
- Desired solver error (first step): This is the desired error, smaller is more accurate and slower. I would generally not accept answers above 1×10^{-5}
- Max Electrical iterations: Maximum number of electrical iterations on all but the first step.
- Electrical clamp: Electrical clamp (first step): This is a number by which the maximum newton step is clamped to. 0.1 will make the solver very stable but very slow, 4.0 will make the solver very fast but unstable. A recommended value of 1.0 is suggested for normal problems. If you are solving for high doping or other unusual conditions it can be worth reducing the step. Likewise if you want the solver to be fast and you know the problem is easy set the value to 2.0 or higher.
- Desired solver error: This is the desired error, smaller is more accurate and slower. I would generally not accept answers above 1×10^{-5}
- Newton solver clever exit: If the solver starts bouncing in the noise then assume we can't get a better answer and quit.
- Newton minimum iterations: Don't allow the solver to quit before doing this number of steps. Often the error in the first few steps of the solution can be below "Desired solver error", thus the solver can quit before finding the true answer.
- Solve Kirchhoff's current law in Newton solver: Solve Kirchhoff's current law in the main Newton Jacobian.
- Matrix solver: This selects the matrix solver to use.
- Newton solver to use:
 - none: No electrical solver is selected, this is used when only solving optical or thermal problems.

- newton: The standard 1D Newton solver.
 - newton_2D: The standard 2D Newton solver.
 - newton_norm: The standard 1D Newton solver but with Slotboom normalization. This is handy when solving systems with large difference in density between minority and majority carrier density.
 - poisson_2d: A 2D Poisson solver with no drift diffusion equations.
- Complex matrix solver:
 - Slotboom T0: Slotboom variable for the newton_norm solver.
 - Slotboom D0: Slotboom variable for the newton_norm solver.
 - Slotboom n0: Slotboom variable for the newton_norm solver.
 - Use newton cache (experimental): Cache large problems to disk - experimental.
 - Quit on convergence problem: Quit on convergence problem. Quite often
 - Quit on inverted Fermi-level:
 - Solver output verbosity:

8.8.1 Solver stability

Avoiding very big and very small numbers

Try opening up MATLAB (Octave if you are on Linux) and typing in the following equation $((1e - 1 + 1e1) - 1e1)/1e - 1$. Before pressing enter, try to evaluate it in your head. the $1e1$ and the $-1e1$ cancel leaving $\frac{1e-1}{1e-1}$ which equates to 1. Now try replacing the powers to 1 with to the 19, so type in $((1e - 19 + 1e19) - 1e19)/1e - 19$, again evaluate this in your head. Again , $1e19$ and the $-1e19$ cancel leaving $\frac{1e-19}{1e-19}$ which equates to 1 Now let the computer evaluate the expression. In fact this time the computer does not give you 1 but gives you 0. Double check that you typed it in correctly... you did so what is happening. Why is the computer giving me an answer which is 100% wrong. The answer is easy, computers have a limited precision. This means that they can only store a limited number of decimal places. On a modern PC it's about 15 decimal places. After this the computer starts ignoring the numbers. So when we added $(1e - 19 + 1e19)$ the computer could not keep track of the decimal places so it assumed that the answer was exactly $1.0000000000000000e19$ and not $1.000000000000000001e19$, then when we subtracted $-1e19$ from the answer the computer gave us zero instead of $1e - 19$. The $1e - 19$ was lost in the precision.

All computers are affected by this no matter how powerful they are, this has important implications when solving device equations. If you have too big a spread of numbers in your simulation (matrix/Jacobian) the computer won't be able to solve it easily. So if you have very low values of mobility say $1e - 19$ and very big values say $1e5$ the computer will start to have problems solving the electrical problem. There fore generally try to reduce the spread of parameters in you model. This is important when simulating insulators.

Avoid zeros

Zeros are bad because they cause divide by zero errors. So don't have zero mobilities, carrier cross sections, tail slopes or densities of states. It's fine to have zero recombination constants though.

Very big steps in the band gap

Big steps in the band gap will produce very small and very large carrier densities - see *Avoiding very big and very small numbers* above.

8.8.2 Simulating disordered devices without traps

This section needs to be rewritten, to more generally talk about recombination and not just Langevin recombination. For a more complete view watch the video below

Related YouTube videos:



Please stop simulating disordered semiconductors without trap states.

In my view Langevin recombination is in general a really bad way to describe recombination in OPV devices. This is because the mechanism assumes Brownian motion of electrons and holes and that charge carriers of opposite polarity will recombine when they get close enough to fall into each others electrostatic field. This picture assumes the charge carriers are free and completely neglects the influence of trap states. I therefore think Langevin recombination should be avoided in OPVs. But in dx.doi.org/10.1021/jp200234m you used Langevin recombination - why?: In this paper I allowed the mobility in the Langevin expression to vary as a function of carrier density i.e.

$$R_{free} = qk_r \frac{(\alpha\mu_e(n) + \beta\mu_h(n))n_{tot}p_{tot}}{2\epsilon_0\epsilon_r} \quad (8.28)$$

I then by defining a mobility edge and assuming any carrier below the mobility edge could not move and any carrier above it could. I could define the averaged electron/hole mobility as:

$$\mu_e(n) = \frac{\mu_e^0 n_{free}}{n_{free} + n_{trap}} \quad (8.29)$$

and

$$\mu_h(n) = \frac{\mu_h^0 p_{free}}{p_{free} + p_{trap}} \quad (8.30)$$

and if one assumes the density of free charge carriers is much smaller than the density of trapped charge carriers one can arrive at

$$R(n, p) = qk_r \frac{(\alpha\mu_e^0 n_{free} p_{trap} + \beta\mu_h^0 p_{free} n_{trap})}{2\epsilon_0\epsilon_r} \quad (8.31)$$

Thus by making the mobility carrier density dependent we arrive at an expression for Langevin recombination that's dependent upon the density of free and trapped carriers (i.e. $n_{free}p_{trap}$ and $p_{free}n_{trap}$) This is in principle the same as SRH recombination (i.e. a process involving free electrons (holes) recombining with trapped holes (electrons)). This was a nice simple approach and it worked quite well in the steady state. However, to make this all work I had to assume all electrons (holes) at any given position in space had a single quasi-Fermi level, which meant they were all in equilibrium with each other. For this to be true, all electrons (holes) would have to be able to exchange energy with all other electrons (holes) at that position in space and have an infinite charge carrier thermalization velocity. This seemed like an OK assumption in steady state when electrons (holes) had time to exchange energy, however once we start thinking about things happening in time domain, it becomes harder to justify because

there are so many trap states in the device it is unlikely that charge carriers will be able to act as one equilibrated gas with one quasi-Fermi level. On the other hand the SRH mechanism does not make this assumption, so it is probably a better description of recombination/trapping. I would also add that I have never found a situation in OPV device modeling where SRH recombination was unable to describe the device in question. Conclusion: SRH is better than Langevin.

8.9 Calculating the built in potential

The first step to performing a device simulation, is to calculate the built in potential of the device. To do this we must know the following things:

- The majority carrier concentrations on the contacts n and p .
- The effective densities of states N_{LUMO} and N_{HOMO} .
- The effective band gap E_g

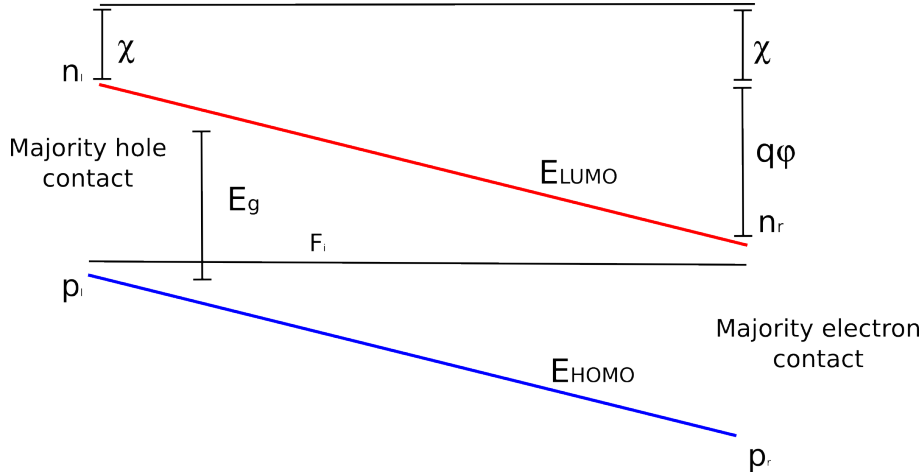


Figure 8.2: Band structure of device in equilibrium.

The left hand side of the device is given a reference potential of 0 V. See figure 8.2. We can then write the energy of the LUMO and HOMO on the left hand side of the device as:

$$E_{LUMO} = -\chi \quad (8.32)$$

$$E_{HOMO} = -\chi - E_g \quad (8.33)$$

For the left hand side of the device, we can use Maxwell-Boltzmann statistics to calculate the equilibrium Fermi-level (F_i).

$$p_l = N_v \exp\left(\frac{E_{HOMO} - F_p}{kT}\right) \quad (8.34)$$

We can then calculate the minority carrier concentration on the left hand side using F_i

$$n_l = N_c \exp\left(\frac{F_n - E_{LUMO}}{kT}\right) \quad (8.35)$$

8.9. CALCULATING THE BUILT IN POTENTIAL

The Fermi-level must be flat across the entire device because it is in equilibrium. However we know there is a built in potential, we can therefore write the potential of the conduction and valance band on the right hand side of the device in terms of ϕ to take account of the built in potential.

$$E_{LUMO} = -\chi - q\phi \quad (8.36)$$

$$E_{HOMO} = -\chi - E_g - q\phi \quad (8.37)$$

we can now calculate the potential using

$$n_r = N_c \exp\left(\frac{F_n - E_{LUMO}}{kT}\right) \quad (8.38)$$

equation 8.36.

The minority concentration on the right hand side can now also be calculated using.

$$p_r = N_v \exp\left(\frac{E_v - F_{HOMO}}{kT}\right) \quad (8.39)$$

The result of this calculation is that we now know the built in potential and minority carrier concentrations on both sides of the device. Note, infinite recombination velocity on the contacts is assumed. I have not included finite recombination velocities in the model simply because they would add four more fitting parameters and in my experience I have never needed to use them to fit any experimental data I have come across.

Once this calculation has been performed, we can estimate the potential profile between the left and right hand side of the device, using a linear approximation. From this the charge carrier densities across the device can be guessed. The guess for potential and carrier densities, is then used to prime the main Newton solver. Where the real value are calculated. The Newton solver is described in the next section.

8.9.1 Average free carrier mobility

In this model there are two types of electrons (holes), free electrons (holes) and trapped electrons (holes). Free electrons (holes) have a finite mobility of μ_e^0 (μ_h^0) and trapped electrons (holes) can not move at all and have a mobility of zero. To calculate the average mobility we take the ratio of free to trapped carriers and multiply it by the free carrier mobility.:

$$\mu_e(n) = \frac{\mu_e^0 n_{free}}{n_{free} + n_{trap}} \quad (8.40)$$

Thus if all carriers were free, the average mobility would be μ_e^0 and if all carriers were trapped the average mobility would be 0. It should be noted that only μ_e^0 (μ_h^0) are used in the model for computation and $\mu_e(n)$ is an output parameter.

The value of μ_e^0 (μ_h^0) is an input parameter to the model. This can be edited in the electrical parameter editor. The value of $\mu_e(n)$, and $\mu_h(p)$ are output parameters from the model. The value of $\mu_e(n)$, and $\mu_h(p)$ change as a function of position, within the device, as the number of both free and trapped charge carriers change as a function of position. The values of $\mu_e(x)$, and $\mu_h(x)$ can be found in *mu_n_ft.dat* and *mu_p_ft.dat* within the *snapshots* directory. The spatially averaged value of mobility, as a function of time or voltage can be found in the files *dynamic_mue.dat* or *dynamic_muh.dat* within the *dynamic* directory.

Were one to try to measure mobility using a technique such as CELIV or ToF, one would expect to get a value closer to $\mu_e(n)$ or $\mu_h(p)$ rather than closer to μ_e^0 or μ_h^0 . It should be

noted however, that measuring mobility in disordered materials is a difficult thing to do, and one will get a different experimental value of mobility depending upon which experimental measurement method one uses, furthermore, mobility will change depending upon the charge density profile within the device, and thus upon the applied voltage and light intensity. To better understand this, try for example doing a CELIV simulation, and plotting $\mu_e(n)$ as a function of time (Voltage). You will see that mobility reduces as the negative voltage ramp is applied, this is because carriers are being sucked out of the device. Then try extracting the mobility from the transient using the CELIV equation for extracting mobility. Firstly, the CELIV equation will give you one value of mobility, which is a simplification of reality as the value really changes during the application of the voltage ramp. Secondly, the value you get from the equation will almost certainly not match either μ_e^0 or any value of $\mu_e(n)$. This simply highlights, the difficult of measuring a value of mobility for a disordered semiconductor and that really when we quote a value of mobility for a disordered material, it really only makes sense to quote a value measured under the conditions a material will be used. For example, for a solar cell, values of $\mu_e(n)$ and $\mu_h(n)$, would be most useful to know under 1 Sun at the P_{max} point on a JV curve.

8.10

There are three options for thermal simulation in OghmaNano; 1) A constant temperature through the device. This is recommended for most simulation and is set at 300K by default; 2) a lattice thermal solver 9.1.1, this solves the heat equation throughout the device taking into account self heating. This is useful for simulating devices which get hot through their operation; 3) A hydrodynamic thermal 9.1.2 solver which does not assume the electron, hole and lattice temperatures are equal. This is useful for simulating heat flow over heterojunctions or where carriers do not have time to relax to the lattice temperature.

The drift diffusion equations given in 8.20 and 8.24 are only valid in isothermal conditions. The full transport equations as derived from the BTE [12] are given by

$$\mathbf{J}_n = \mu_e n \nabla E_c + \frac{2}{3} \mu_e n \nabla \bar{W} + \frac{2}{3} \bar{W} \mu_e \nabla n - \mu_e n \bar{W} \frac{\nabla m_e^*}{m_e^*} \quad (8.41)$$

$$\mathbf{J}_p = \mu_h p \nabla E_v - \frac{2}{3} \mu_h p \nabla \bar{W} - \frac{2}{3} \bar{W} \mu_h \nabla p + \mu_h p \bar{W} \frac{\nabla m_h^*}{m_h^*} \quad (8.42)$$

where \bar{W} is the average kinetic energy of the free carriers as given by 8.10. If the average energy is assumed to be $3/2kT$, 9.1 and 9.2, return to the standard drift diffusion equations. Note the full form of these equations is required when not using MB statistics.

The thermal model can be configured in the thermal ribbon 9.1. Usually the thermal model is turned off and a constant temperature (300K) is assumed across the device. If you wish to adjust this temperature click on the "Set temperature icon". The thermal model can be turned on by clicking on the candle to the on the far left of the thermal ribbon, so that a flame appears. Various heating sources can be enabled or disabled by depressing the buttons to the right of the ribbon. Boundary conditions can be set in the "Boundary Conditions" window, thermal constants of the material layers can be changed in the "Thermal parameters window".

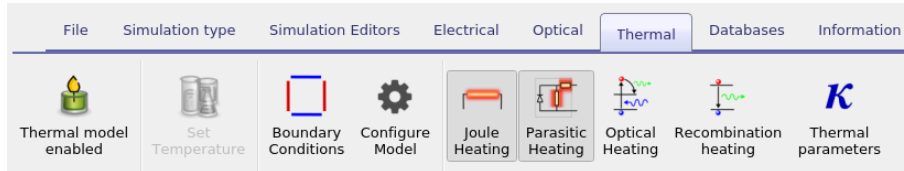


Figure 8.3: Thermal

8.10.1 Lattice thermal model

When solving only the lattice heat equation heat transfer and generation is given by

$$0 = \nabla \kappa_l \nabla T_L + H_j + H_r + H_{optical} + H_{shunt} \quad (8.43)$$

where joule heating (H_j) is give by

$$H_j = J_n \frac{\nabla E_c}{q} + J_h \frac{\nabla E_h}{q}, \quad (8.44)$$

recombination heating (H_r) is given by,

$$H_r = R(E_c - E_v) \quad (8.45)$$

optical absorption heating is given by,

$$H_{optical} \quad (8.46)$$

and heating due to the shunt resistance is given by

$$H_{shunt} = \frac{J_{shunt} V_{applied}}{d}. \quad (8.47)$$

The thickness of the device is given by d . Note shunt heating is only in there to conserve energy conservation.

8.10.2 Energy balance - hydrodynamic transport model

If you turn on the electrical and hole thermal model, then the heat source term will be replaced by

$$H = \frac{3k_b}{2} \left(n \left(\frac{T_n - T_l}{\tau_e} \right) + p \left(\frac{T_p - T_l}{\tau_h} \right) \right) + R(E_c - E_v) \quad (8.48)$$

and the energy transport equation for electrons

$$S_n = -\kappa_n \frac{dT_n}{dx} - \frac{5}{2} \frac{k_b T_n}{q} J_n \quad (8.49)$$

and holes,

$$S_p = -\kappa_p \frac{dT_p}{dx} + \frac{5}{2} \frac{k_b T_p}{q} J_p \quad (8.50)$$

will be solved.

The energy balance equations will also be solved for electrons,

$$\frac{dS_n}{dx} = \frac{1}{q} \frac{dE_c}{dx} J_n - \frac{3k_b}{2} \left(RT_n + n \left(\frac{T_n - T_l}{\tau_e} \right) \right) \quad (8.51)$$

and for holes

$$\frac{dS_p}{dx} = \frac{1}{q} \frac{dE_v}{dx} J_p - \frac{3k_b}{2} \left(RT_p + n \left(\frac{T_p - T_l}{\tau_e} \right) \right) \quad (8.52)$$

The thermal conductivity of the electron gas is given by

$$\kappa_n = \left(\frac{5}{2} + c_n \right) \frac{k_b^2}{q} T_n \mu_n n \quad (8.53)$$

and for holes as,

$$\kappa_p = \left(\frac{5}{2} + c_p \right) \frac{k_b^2}{q} T_p \mu_p p \quad (8.54)$$

Chapter 9

Optical models

9.1

There are three options for thermal simulation in OghmaNano; 1) A constant temperature through the device. This is recommended for most simulation and is set at 300K by default; 2) a lattice thermal solver 9.1.1, this solves the heat equation throughout the device taking into account self heating. This is useful for simulating devices which get hot through their operation; 3) A hydrodynamic thermal 9.1.2 solver which does not assume the electron, hole and lattice temperatures are equal. This is useful for simulating heat flow over heterojunctions or where carriers do not have time to relax to the lattice temperature.

The drift diffusion equations given in 8.20 and 8.24 are only valid in isothermal conditions. The full transport equations as derived from the BTE [12] are given by

$$\mathbf{J}_n = \mu_e n \nabla E_c + \frac{2}{3} \mu_e n \nabla \bar{W} + \frac{2}{3} \bar{W} \mu_e \nabla n - \mu_e n \bar{W} \frac{\nabla m_e^*}{m_e^*} \quad (9.1)$$

$$\mathbf{J}_p = \mu_h p \nabla E_v - \frac{2}{3} \mu_h p \nabla \bar{W} - \frac{2}{3} \bar{W} \mu_h \nabla p + \mu_h p \bar{W} \frac{\nabla m_h^*}{m_h^*} \quad (9.2)$$

where \bar{W} is the average kinetic energy of the free carriers as given by 8.10. If the average energy is assumed to be $3/2kT$, 9.1 and 9.2, return to the standard drift diffusion equations. Note the full form of these equations is required when not using MB statistics.

The thermal model can be configured in the thermal ribbon 9.1. Usually the thermal model is turned off and a constant temperature (300K) is assumed across the device. If you wish to adjust this temperature click on the "Set temperature icon". The thermal model can be turned on by clicking on the candle to the on the far left of the thermal ribbon, so that a flame appears. Various heating sources can be enabled or disabled by depressing the buttons to the right of the ribbon. Boundary conditions can be set in the "Boundary Conditions" window, thermal constants of the material layers can be changed in the "Thermal parameters window".

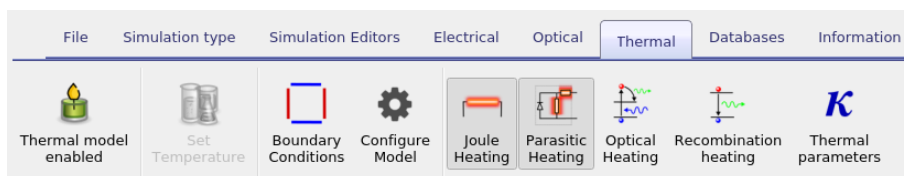


Figure 9.1: Thermal

9.1.1 Lattice thermal model

When solving only the lattice heat equation heat transfer and generation is given by

$$0 = \nabla \kappa_l \nabla T_L + H_j + H_r + H_{optical} + H_{shunt} \quad (9.3)$$

where joule heating (H_j) is give by

$$H_j = J_n \frac{\nabla E_c}{q} + J_h \frac{\nabla E_h}{q}, \quad (9.4)$$

recombination heating (H_r) is given by,

$$H_r = R(E_c - E_v) \quad (9.5)$$

optical absorption heating is given by,

$$H_{optical} \quad (9.6)$$

and heating due to the shunt resistance is given by

$$H_{shunt} = \frac{J_{shunt} V_{applied}}{d}. \quad (9.7)$$

The thickness of the device is given by d. Note shunt heating is only in there to conserve energy conservation.

9.1.2 Energy balance - hydrodynamic transport model

If you turn on the electrical and hole thermal model, then the heat source term will be replaced by

$$H = \frac{3k_b}{2} \left(n \left(\frac{T_n - T_l}{\tau_e} \right) + p \left(\frac{T_p - T_l}{\tau_h} \right) \right) + R(E_c - E_v) \quad (9.8)$$

and the energy transport equation for electrons

$$S_n = -\kappa_n \frac{dT_n}{dx} - \frac{5}{2} \frac{k_b T_n}{q} J_n \quad (9.9)$$

and holes,

$$S_p = -\kappa_p \frac{dT_p}{dx} + \frac{5}{2} \frac{k_b T_p}{q} J_p \quad (9.10)$$

will be solved.

The energy balance equations will also be solved for electrons,

$$\frac{dS_n}{dx} = \frac{1}{q} \frac{dE_c}{dx} J_n - \frac{3k_b}{2} \left(RT_n + n \left(\frac{T_n - T_l}{\tau_e} \right) \right) \quad (9.11)$$

and for holes

$$\frac{dS_p}{dx} = \frac{1}{q} \frac{dE_v}{dx} J_p - \frac{3k_b}{2} \left(RT_p + n \left(\frac{T_p - T_l}{\tau_e} \right) \right) \quad (9.12)$$

The thermal conductivity of the electron gas is given by

9.1.

$$\kappa_n = \left(\frac{5}{2} + c_n \right) \frac{k_b^2}{q} T_n \mu_n n \quad (9.13)$$

and for holes as,

$$\kappa_p = \left(\frac{5}{2} + c_p \right) \frac{k_b^2}{q} T_p \mu_p p \quad (9.14)$$

9.1.3 Ray tracing model

Add text.

Chapter 10

Simple circuit simulations

OghmaNano was primarily designed as a tool to perform detailed device simulations, however sometimes one does not need a full device simulation to understand what is happening in your device. On some occasions a simple circuit model comprising of resistors, capacitors, and ideal diodes will do. For these occasions OghmaNano includes an electrical circuit solver. The circuit solver is a drop in replacement for the drift diffusion solver in that the voltages applied to it are defined in exactly the same way, the experimental modes such as time domain, frequency domain and EQE all work with the circuit solver. Furthermore, the transfer matrix model which is used to calculate how much light is absorbed in each layer can be connected to the diodes, thus enabling photocurrent to be correctly simulated. There are a few examples of circuit simulations in the mode, these can be found in the *Simple Diode Model* folder of the new simulation folder (see figure 10.1.).

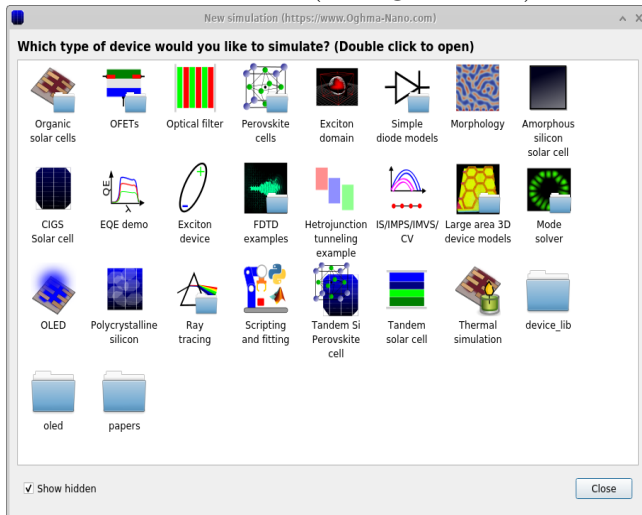


Figure 10.1: Selecting the *Simple circuit simulation example*

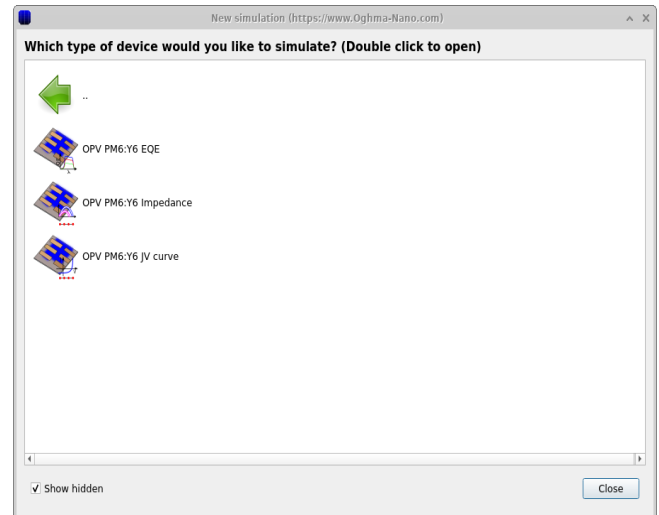


Figure 10.2: Selecting the example that generates the JV curve

With in this folder there are a few example circuit simulations (see Figure 10.2).

If one opens the *OPV PM6:Y6 JV curve* one will get a simulation that looks just like other simulations in OghmaNano (see Figure 10.2), however this simulation has another tab called *circuit diagram* in the main window, if one clicks it one should see a circuit diagram as show in Figure 10.4. This is the circuit diagram editor. On the left is a toolbar, from the top the toolbar provides the following functionality:

- Resistor: This adds a resistor to the circuit.
- Capacitor: This adds a capacitor to the circuit.
- Diode: This adds a standard diode to the circuit of form $i(t, V) = I_0(e^{\frac{qV}{nkT}} - 1) - I_{light}$, I_{light} is taken from the optical simulations.

- Non-linear element: This adds a non-linear circuit element of form $i(t, V) = \frac{I_0 * V^m}{V_0 + d}$
- Wire: A perfect wire with no parasitic parameters.
- Earth: This acts as a ground set at 0V.
- Battery: This applies the voltage to the circuit. The voltage is taken from the contact marked *change* in the contact editor.
- Pointer: This is used to select and edit circuit elements.
- Brush: This is used to delete circuit elements.

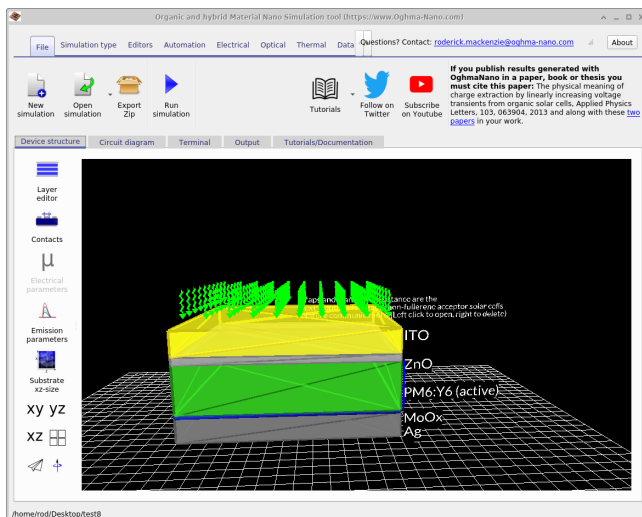


Figure 10.3: The usual interface opens when the circuit simulation is selected.

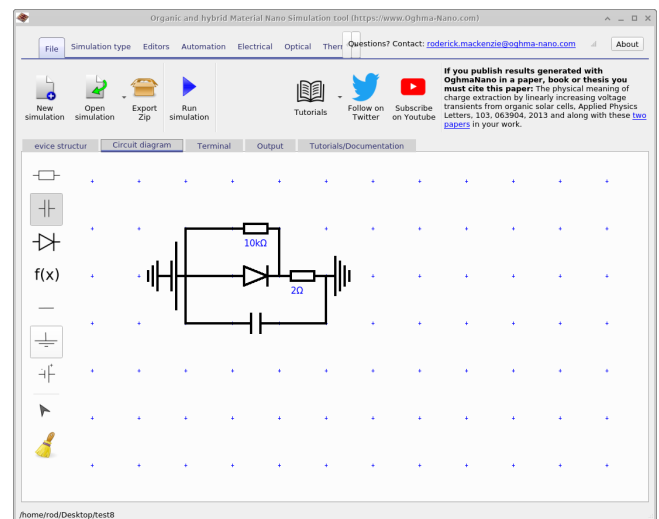


Figure 10.4: The circuit tab of the example showing the circuit diagram used to simulate the device.

By clicking on any circuit element with any tool apart from the brush, you can change the values of the components as seen in Figure 10.5, and zoomed in Figure 10.6. Figure 10.6 shows the configuration window for the diode component. From the top the options are:

- Component: This can be used to change what component the circuit element represents.
- Name: This is an human readable name given to the circuit element, you can call it what you want.
- Ideality factor: The diode ideality factor n .
- I0: Saturation current in the diode equation.
- Layer: This is the layer that the diode represents, the light current will be calculated from the generation in this layer.

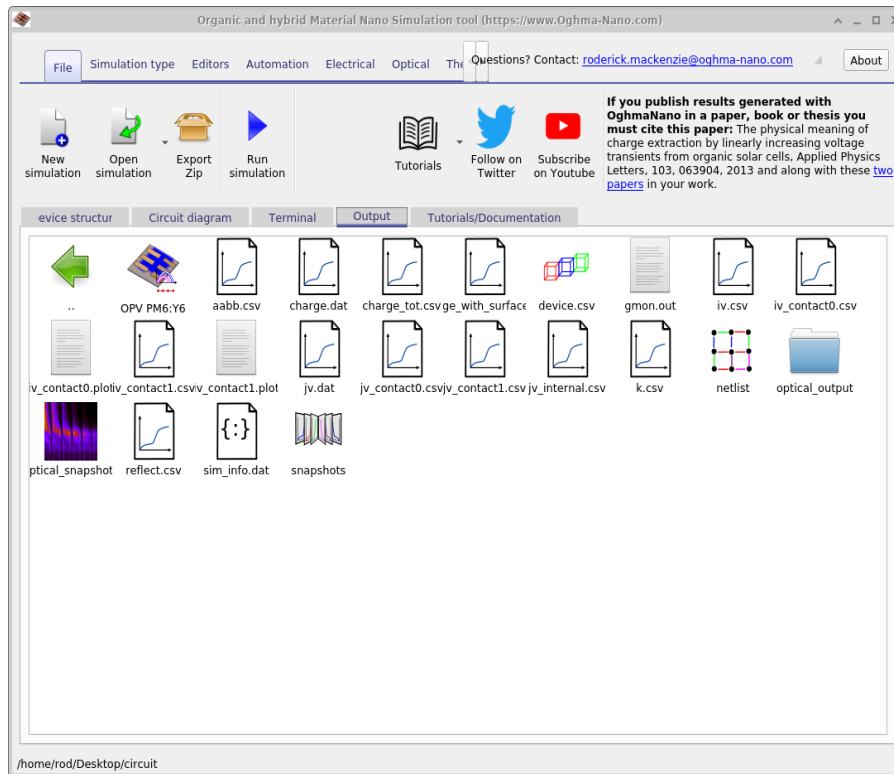


Figure 10.7: The output of circuit simulation window is exactly as it would be for standard drift diffusion simulations.

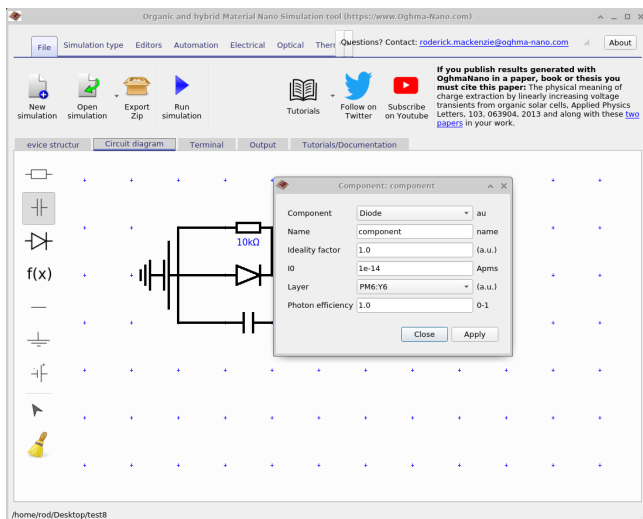


Figure 10.5: Editing the component values of a diode.

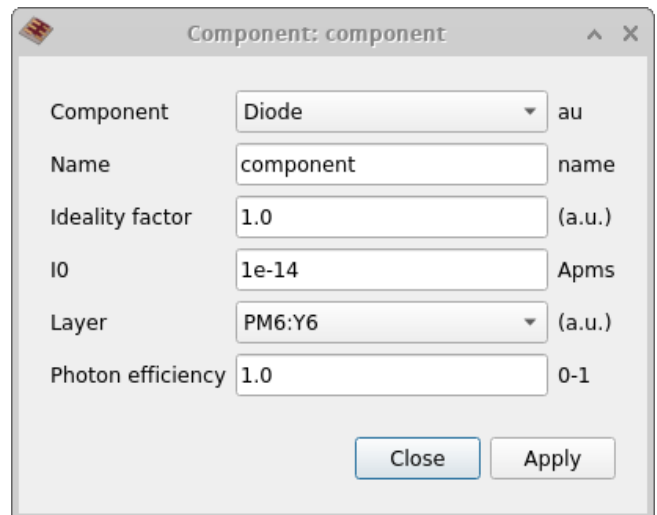


Figure 10.6: A zoomed in view of the diode component editor. Access this menu by clicking on a component.

After you have run the simulation by clicking the play button or by pressing F9, simulation output will be visible in the output tab as usual. All the files you would expect from the usual drift diffusion simulations will be generated. One extra output that is generated in the circuit simulation is the *Net list* this is visible in Figure 10.8, when you double click on this it brings up the Net list window which is visible in Figure 10.8, this shows the voltage over and current through every component in the circuit. You can use the slider to step through the simulation steps, these will be time or voltage steps. The net list is only generated when the simulation output is set to *Write everything to disk* in the simulation editor.

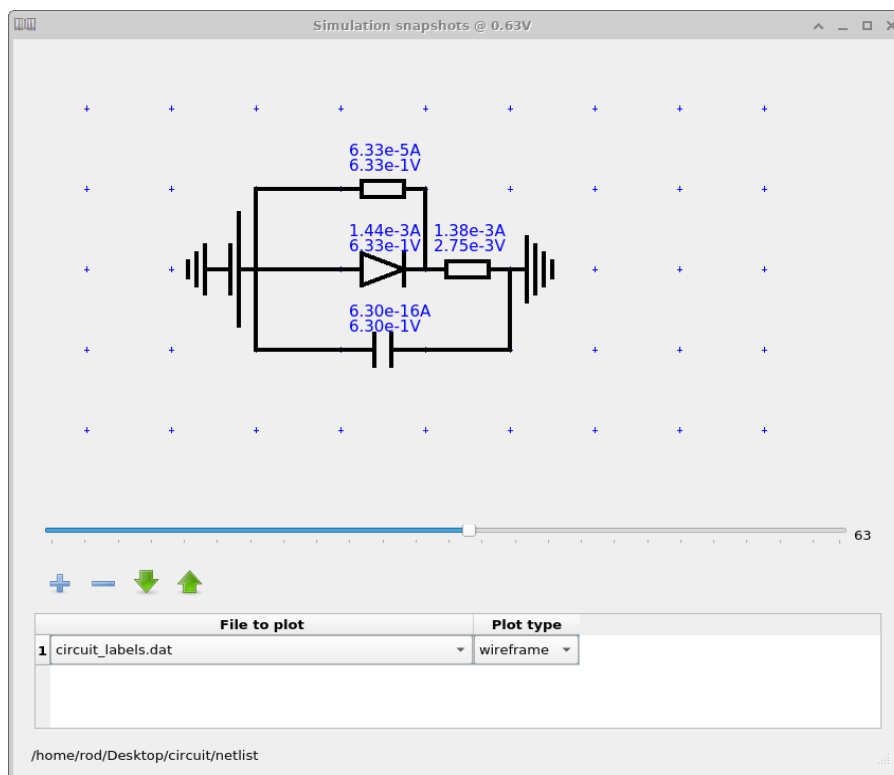


Figure 10.8: The net list, showing the voltages over components and currents through components.

10.0.1 JV, IS, CV and other simulation modes

As mentioned above the circuit simulator is compatible all simulation modes in OghmaNano, by switching the simulation mode to Impedance Spectroscopy one can simulate the frequency response of the circuit (see Figure 10.9), the result of which can be seen in Figure 10.10 where the file `real_imag.csv` has been plotted.

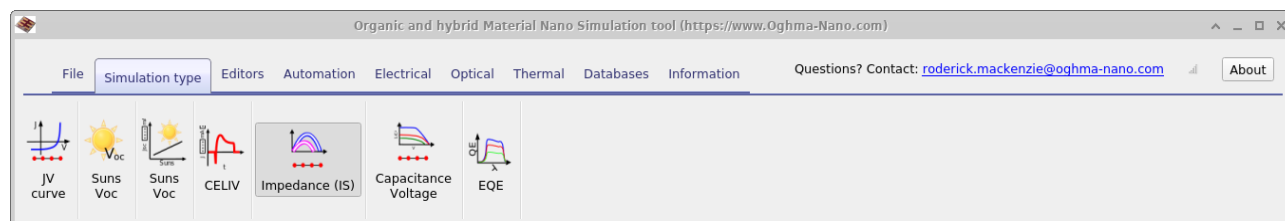


Figure 10.9: Changing the simulation mode to impedance spectroscopy.

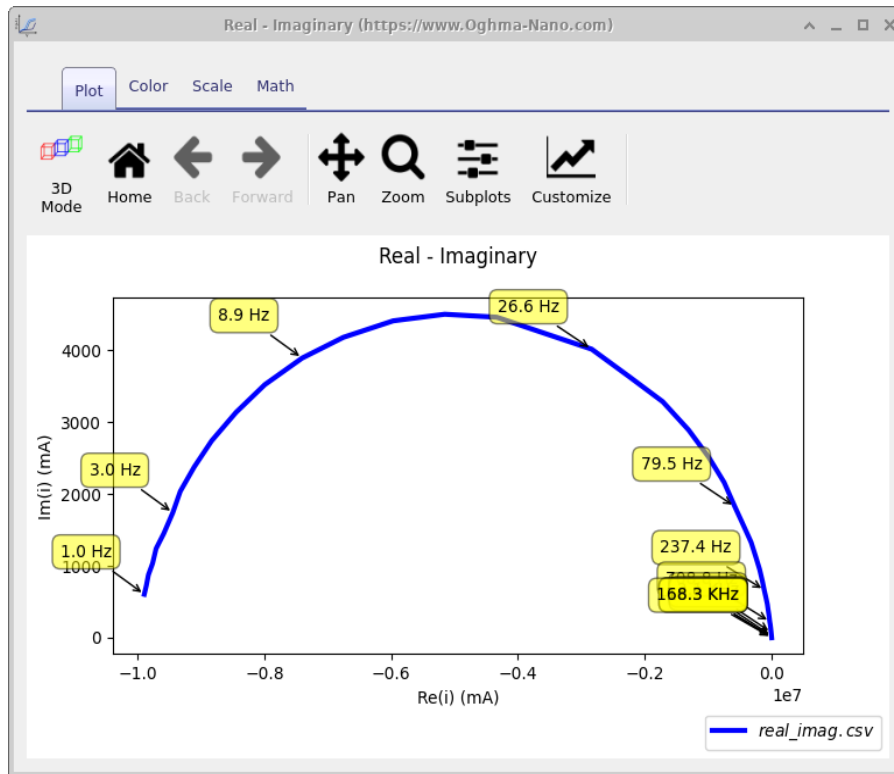


Figure 10.10: An impedance spectroscopy simulation performed using the above circuit. To do this just change the simulation mode to IS.

10.0.2 Using the fitting/scan tools with circuit models

The circuit models are exposed in the json tree just like the drift diffusion material paramters and therefore you can also use the fitting and scan tools to either fit the data to experiment or to scan through circuit values.

Chapter 11

Large area device simulation

OghmaNano primarily focuses on drift diffusion modelling of small area device such as solar cells and OFETs. Drift and diffusion simulations are good at describing the microscopic operation of devices. They allow you to understand how carriers, potential and recombination interact on the nanometer scale. However, sometimes one wants to simulate large area devices such as printed substrates spanning over many square centimetres. For this type of simulation one needs to use less detailed and more efficient circuit models, this section describes how to do that.

Related YouTube videos:



Tutorial on designing large area contacts for flexible electronics



Understanding Printed Hexagonal Contacts for Large Area Solar Cells

11.1 Designing contacts for large area devices

A common problem is designing large area contacts for solar cells. This paper [13] gives an overview of such a problem. To start designing large area contacts open the new simulation window in the file ribbon, and select the *Large area hexagonal contact* simulation (see figure 11.1). Once you have opened it you should get a window which looks like figure 11.2. This simulation consists for a hexagonal solver contact printed on top of a PEDOT substrate. We are going to find out how the resistance of this contact varies as a function of position.

11.1. DESIGNING CONTACTS FOR LARGE AREA DEVICES

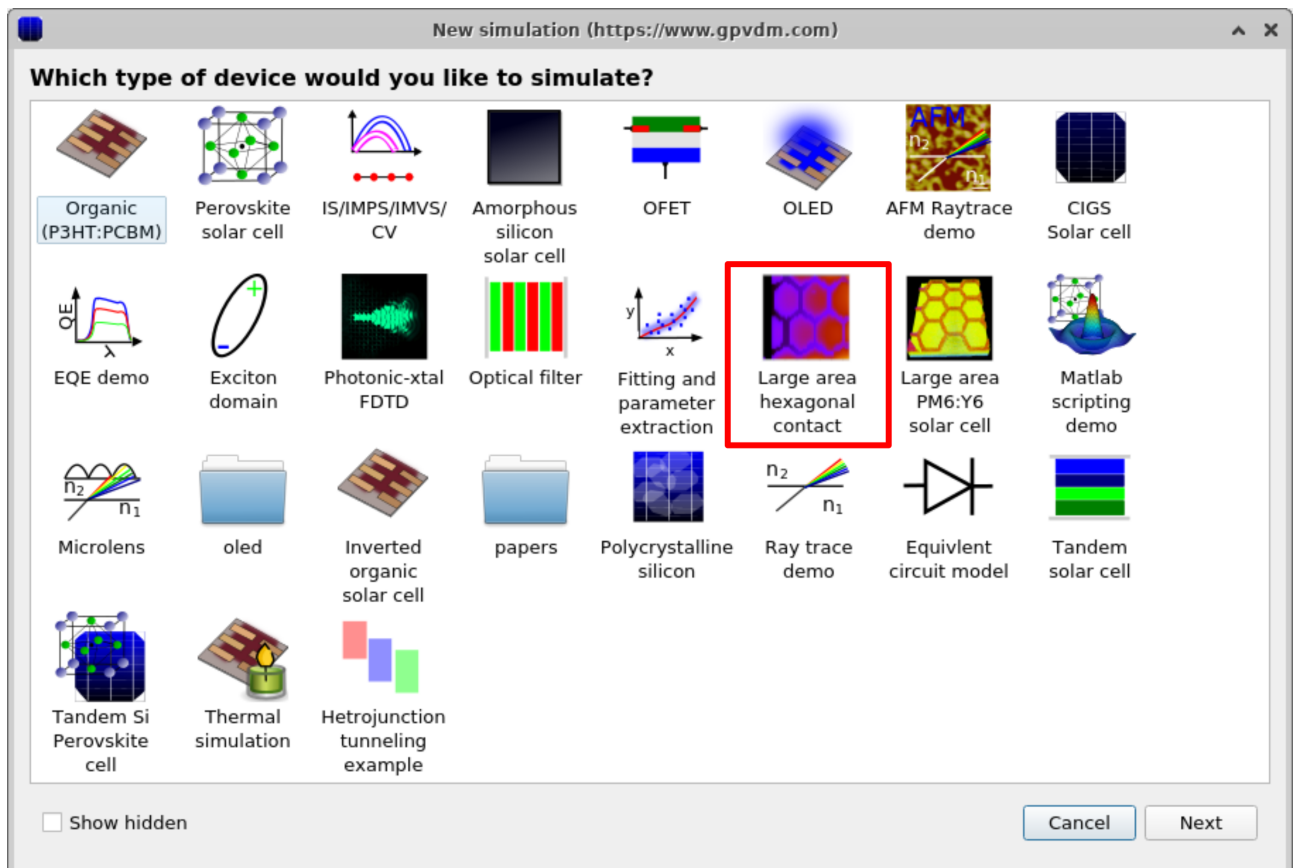


Figure 11.1: Selecting the large area contact simulation

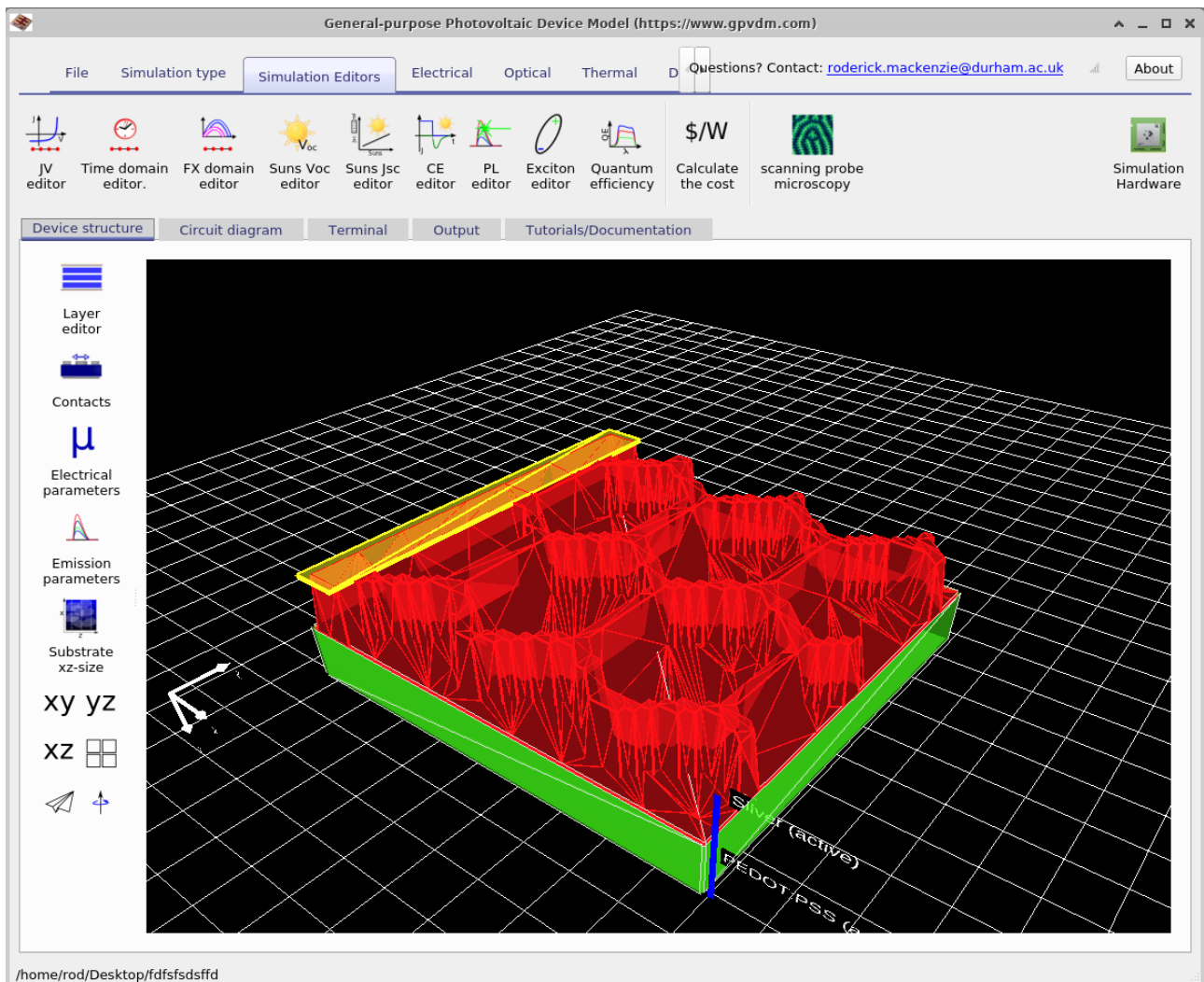


Figure 11.2: A 3D image of the contact printed contact.

The next step in the simulation is to build a network of resistors which approximates the shape of the contact. To do this select the Circuit diagram tab and then click the refresh button. This will build a resistor network of the shape shown in the device structure tab, see figure 11.3. Here you can zoom in and examine the individual resistors, each line represents a resistor.

11.1. DESIGNING CONTACTS FOR LARGE AREA DEVICES

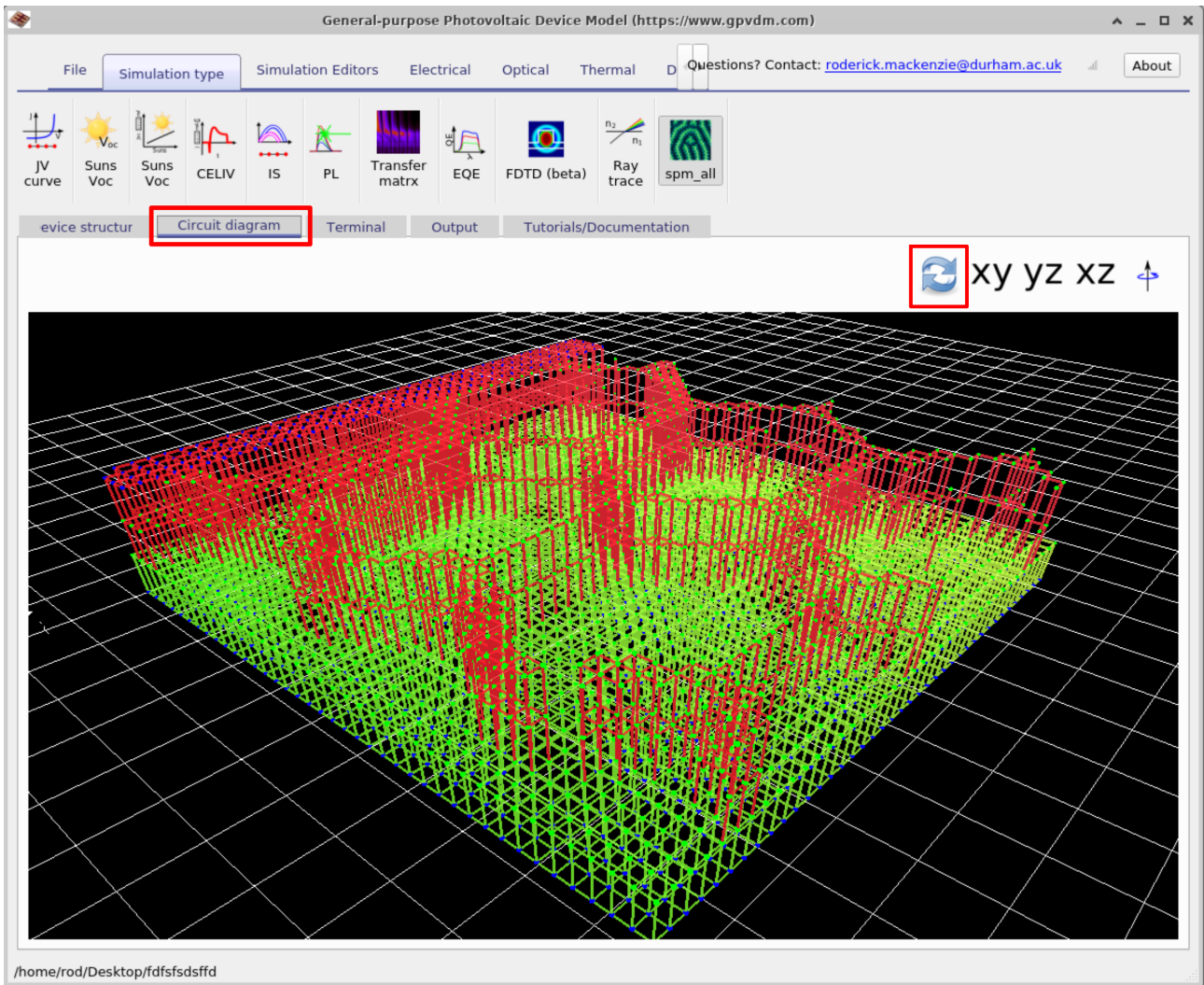


Figure 11.3: Building the 3D circuit mesh of the contact structure.

Once this is built we can run a full simulation and calculate the resistance between the bottom of the PEDOT:PSS layer (bottom of the green layer in figure 11.2) and the extracting silver contact (far left yellow strip on the top of figure 11.2). Run the simulation by clicking on the play button in the file ribbon.

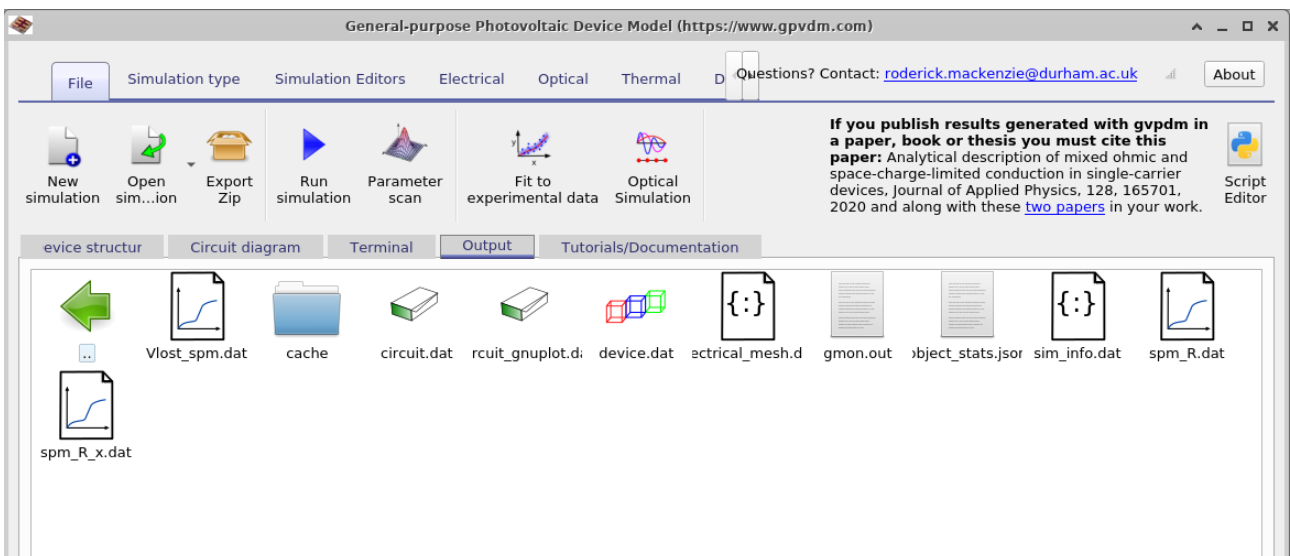


Figure 11.4: A 1D diagram of the mesh

The simulation may take a while to run, once it has finished you can open the output files in the *Output* tab, see figure ???. If you open the file called *spm_R.dat* it will show you a resistance map of the structure which can be seen in figure 11.5. Other output files are listed below in table 11.1.

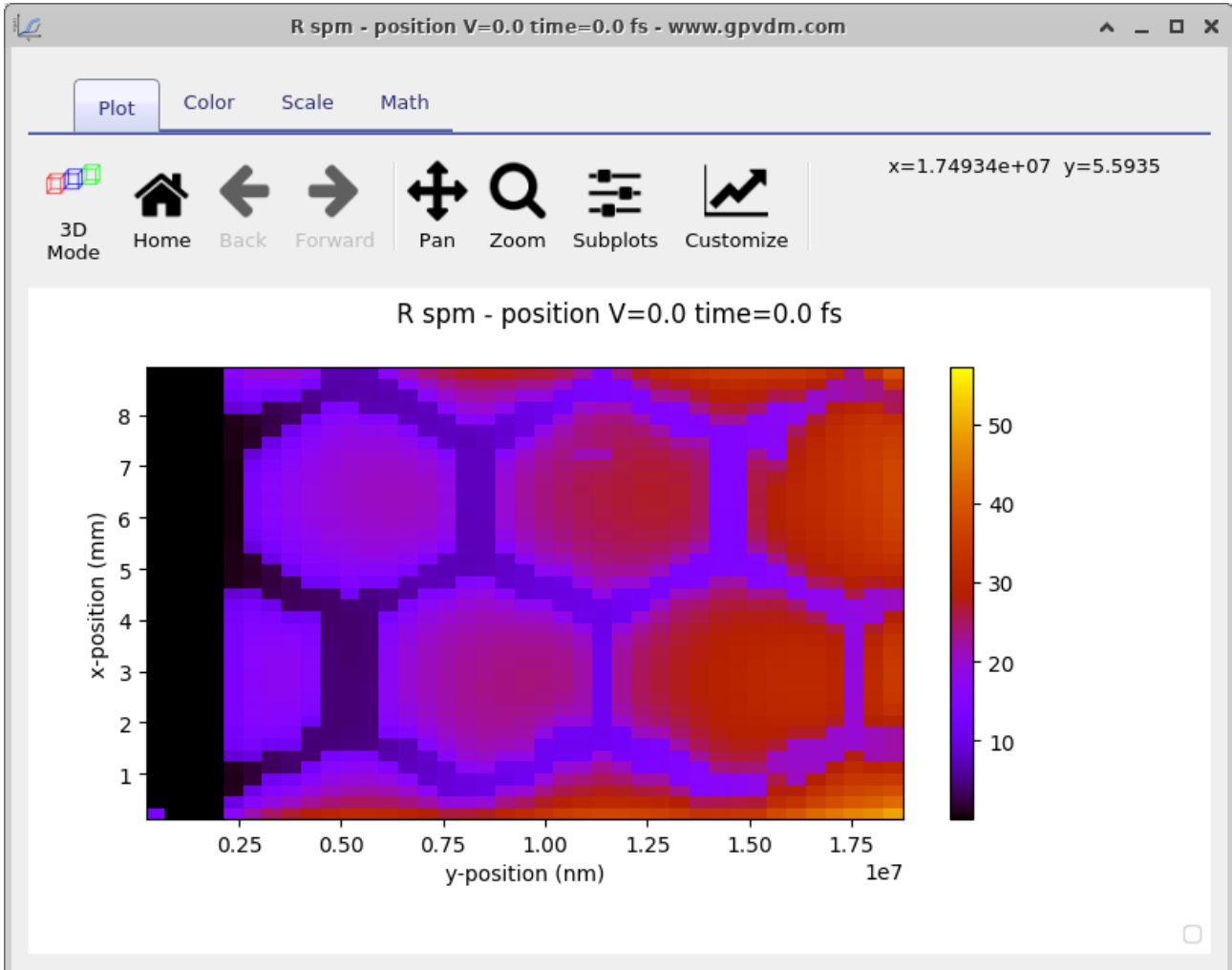


Figure 11.5: A 2D resistance plot across the surface of the device.

File name	Description
<i>spm_R.dat</i>	2D plot of resistance
<i>spm_R_x.dat</i>	A resistance plot down the centre of the device.

Table 11.1: Files produced by the SPM simulator.

11.2. SIMULATING LARGE AREA SOLAR CELLS

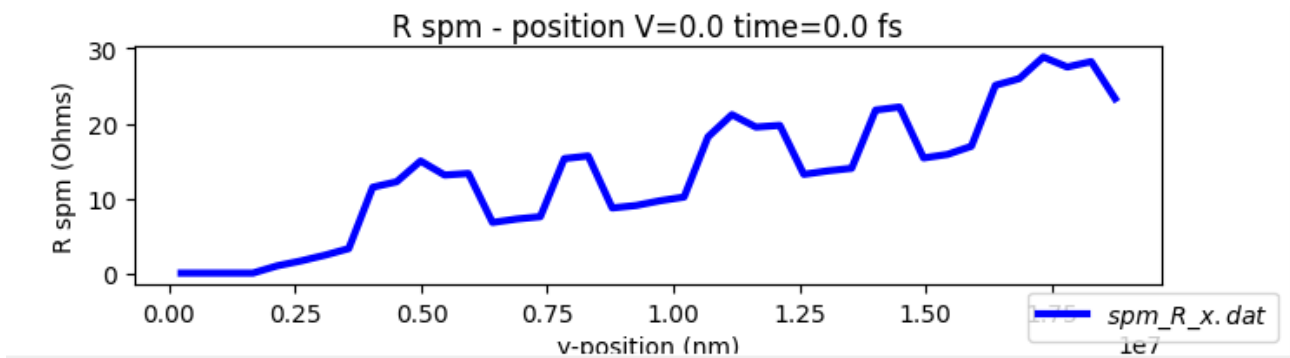


Figure 11.6: A 1D resistance plot taken through the centre of the device.

The scanning probe microscopy editor can be found in the *Simulation Editors* ribbon in the main window. This can be used to select if one scans the entire device or only section of it. The editor can be seen in figure 11.7

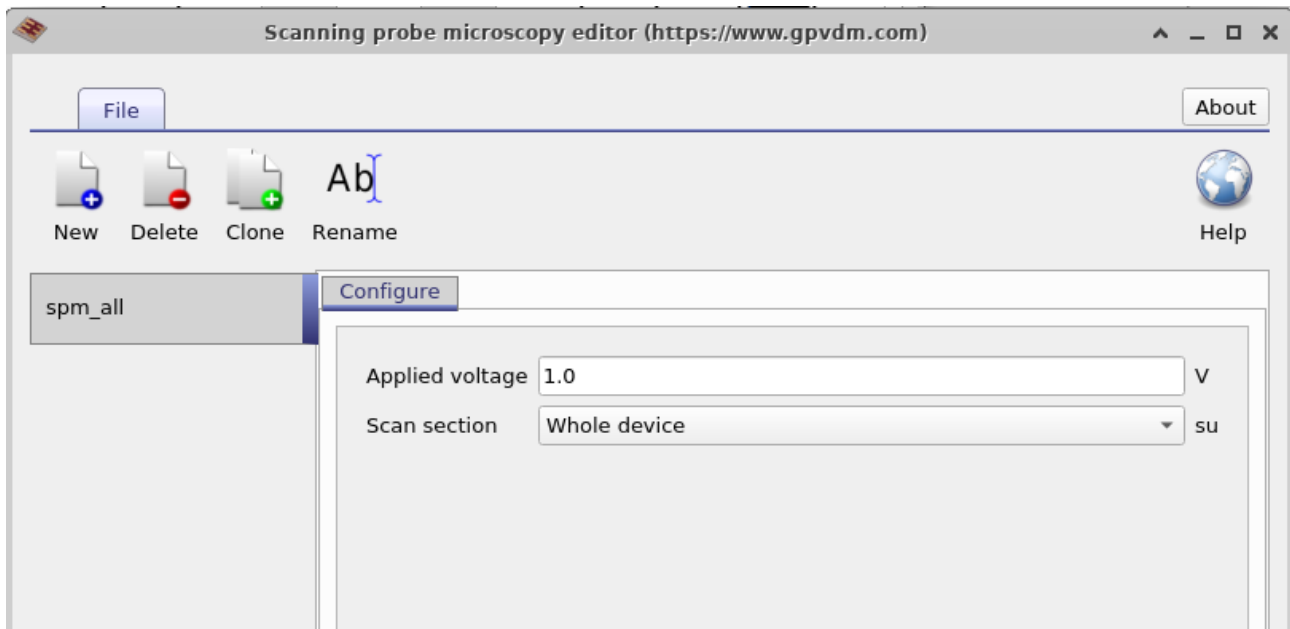


Figure 11.7: The output files from the simulation.

11.2 Simulating large area solar cells

Chapter 12

Modelling excitons/geminate recombination - organics only

12.1 Why you should not model excitons

There are a number of models to calculate the number of geminate pairs which get converted to free charge carriers the Onsager-Braun model for example will give you the exciton dissociation efficiency. There are other models which will enable you to calculate the distribution of excitons in a device as a function of position. However, these models will generally require a number of parameters which are often not reliably known for a material system. Such parameters include exciton life-time, diffusion length and dissociation rate. So although it's possible (and interesting) to write a model to simulate geminate recombination, one is usually better off simply introducing a *photon efficiency factor* η_{photon} . This number ranges between 0.0 and 1.0 and is multiplied by the number of photons absorbed at any point in the device to account for geminate recombination losses.

$$G = G_{abs} \cdot \eta_{photon} \quad (12.1)$$

where G is the charge carrier generation rate in $m^{-3}s^{-1}$ in equations 12.1 and 8.25.

This factor can be obtained to a reasonable degree by comparing the difference between the simulated and experimental J_{sc} . This parameter can set in the configuration section of the optical simulation window. So therefore my advice is that in most cases you should not be modelling excitons explicitly but rather using the 'photon efficiency factor'. If you really want to model excitons read on..

12.2 Modelling excitons

So if you have read section 12.1 and still think you want to model excitons this section will explain how to do it. Gvpdm includes an exciton solver. This sits between the optical model and electrical model. If the exciton model is turned off then generation is simply the number of photons absorbed at any point in the device multiplied by the *photon efficiency factor* see equation 12.1. If the exciton model is turned on then optical absorption will feed straight into the exciton diffusion equation.

$$\frac{\partial X}{\partial t} = \nabla \cdot D \nabla X + G_{optical} - k_{dis}X - k_{FRET}X - k_{PL}X - \alpha X^2. \quad (12.2)$$

where X is the exciton density as a function of position, D is the diffusion constant, $G_{optical}$ exciton generation rate. This value is taken straight from the optical model. The constant k_{dis}

12.3. MODELING EXCITATIONS IN A DEVICE

is exciton dissociation rate to free charge carriers. When the exciton model is switched on G in equations equals $k_{dis}X$. k_{FRET} is the Förster resonance energy transfer, $k_{PL}X$ is the radiative loss and α is an exciton-exciton annihilation rate constant. The diffusion term is defined as

$$D = \frac{L^2}{\tau} \quad (12.3)$$

Where L is exciton diffusion length and τ is the exciton lifetime.

12.3 Modeling excitations in a device

12.4 Modeling excitations in a unit cell

Chapter 13

The oghma file format

13.1 the .oghma simulation file format

In OghmaNano simulations are saved in a directory containing a sim.oghma file. All the parameters specifying the device and simulations are stored in the sim.oghma file. If you rename the file so to be called sim.zip you will be able to open it in windows explorer or your favourite zip viewer. Inside the .oghma file you will find another file called sim.json. You can view this file in any text editor but the file is quite long so I recommend you use firefox as it has a very nice built in json viewer. Json is a simple way of storing text and configuration information first developed for Java. Json is a standard way to store and transmit data much like XML. You can see examples here: <https://json.org/example.html> or below in code listing 1.

You can see the json file is structured using a series of brackets, double quotes and commas. If you make a copy of sim.json outside the .oghma archive, then rename the sim.zip back to sim.oghma, OghmaNano will ignore the sim.json file within the sim.oghma archive and revert to the plain text file stored in the simulation directory. This feature can be useful for automation of simulations as you can simply edit the sim.json file using your favourite programming language without having to learn about reading and writing zip files. If you open the sim.json file in firefox it will look like 13.2. Also have a look at the file in notepad to get a sense of what is in it.

You can see that the json file has various headings, key headings are listed below in table 13.1. If you wish to programmatically drive OghmaNano you can simply use one of the many available json editors most languages have them freely available.

```
{  
  "color_of_dog": "brown",  
  "dog_age": 5,  
  "dogs_toys": {  
    "rabbit": "True",  
    "stick": "False"  
  }  
}
```

Figure 13.1: A simple JSON example

13.2 Qwerks of the OghmaNano json format

- The OghmaNano json file does not support standard json lists e.g. ["Red", "Green", "Blue"]. If there is a list of items, it is defined by firstly declaring the variable segments, with the number of items in the list so for example "segments",0 . Each item in the list is then stored under, "segment0", "segment1" etc... This format enables OghmaNano to allocate the memory for reading in the structures before doing the reading. This can be

13.2. QWERKS OF THE OGHMANANO JSON FORMAT



Figure 13.2: An example sim.json file opened in Firefox.

Heading	Description
sim	General simulation information
jv	JV curve configuration
dump	Defines how much information is written to disk
math	Math configuration for the solver
light	Optical transfer matrix configuration
light_sources	Configuration of light sources
epitaxy	Defines the structure of the device
thermal	Thermal configuration
thermal_boundary	Thermal boundary config.
exciton	Exciton config
exciton_boundary	Exciton boundary config.
ray	Ray tracing config.
suns_voc	Suns-Voc
suns_jsc	Suns-Jsc
ce	Charge Extraction config.
transfer_matrix	Light transfer matrix config
pl_ss	PL in steady state
eqe	EQE config.
fdtd	FDTD config.
fits	Fitting config.
mesh	Electrical mesh config.
time_domain	Time domain config.
fx_domain	FX-domain config
cv	CV config.
parasitic	Parasitic components
spm	Scanning Probe Microscopy config.
hard_limit	Setting hard limits for sim params.
perovskite	Perovskite solver config.
electrical_solver	Electrical solver config.
spctral2	SPCTRAL2
lasers	fs Lasers
circuit	Circuit solver config.
gl	OpenGL config
world	Defines the world box

Table 13.1: Key headings/sections in the sim.json file.

13.3. ENCODING

seen in figure 13.2 where there is a list with 1 segment.

- Many items in the json file will be given an ID number which is a 16 digit hex code, this can be used to uniquely reference the item. An ID number can also be seen in figure 13.2. These ID numbers are generated at random but every ID number must be unique. ID numbers enable objects for example epitaxy layers to be identified uniquely even if they have the same name.

13.3 Encoding

The .json files read/written by OghmaNano are always stored in UTF-8 format. OghmaNano can not handle UTF-16 or any other text encoding standards. Nowadays windows notepad and most other apps default to UTF-8, so if you don't know what these text storage formats are it probably does not matter. This will only rear it's head if you start programmatically generating .oghma files in a language such as C++ and are using a language such as Chinese or Russian with non Latin characters in it's alphabet.

13.4 Forwards/backwards compatability of the file format

Significant effort is made to make sure new versions of OghmaNano can read files generated in older versions. However, older versions of OghmaNano may not be able to read files generated on newer versions. Every time the user opens a sim.oghma file using the GUI the file format is checked and if it differs to that being used in the current version the file is updated and written back to disk. If you are using OghmaNano in a headless configuration by calling *oghma_core.exe* directly, then when sim.oghma files from old versions of the model, before running *oghma_core.exe*, make sure you have opened it in the GUI first to make sure the file is in the correct format.

Chapter 14

Databases

There are a series of databases used to define material parameters, shapes, emission spectra and solar spectra etc... These are described within this section. From the graphical user interface they can be accessed from the database ribbon, see figure 14.1.

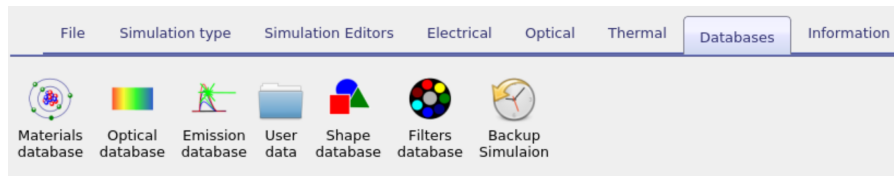


Figure 14.1: The database ribbon

There are two copies of these databases, one copy in the install directory of `OghmaNanoC:\Program Files\OghmaNano\` and one in your home directory in a folder called `OghmaNano_local`. When the model starts for the first time it copies the read only materials database from, to the `OghmaNano_local` folder in your home directory. If you delete the copy of the materials database in the `OghmaNano_local` folder it will get copied back next time you start the model, this way you can always revert to the original databases if you damage the copy in `OghmaNano_local`.

The structure of the databases are simple, they are a series of directories with one directory dedicated to each material or spectra etc.. E.g. there will be one directory called `Ag` in the optical database which defines silver, and another directory in the spectra database called `am1.5g` which defines the solar spectrum. Each of these database directories will from now on be referred to an object. Within each object there is a `data.json` file which defines basic material properties and configuration information. There will may be a couple of `.bib` files which contain reference information for the object in bibtex format and either `.gmat` files for n/k spectral data or `.inp` files for other types of data. All these files are just human readable text files, so you can open them in your preferred text editor such as notepad.

14.1 Materials database

Related YouTube videos:



A tutorial on adding new materials to gvpdm

This database primarily contains n/k data but also contains some electrical information and thermal information. Each subdirectory within the materials database identifies the material name. In each sub directory there are two key files *alpha.gmat* and *n.gmat*, these files are standard text files can be opened with any text editor such as wordpad. Alpha.gmat contains the absorption coefficient of the material while n.gmat contains the the refractive index. The first column of the file contains the wavelength in *m* (not *cm* or *nm*), and the second column of the file contains the absorption coefficient in m^{-1} (for alpha.omat) and the real part of the refractive index (i.e. n) in au (for n.omat). The data.json defines the material color and any known electrical or thermal data.

14.2 Adding new materials - the hard way

If you wish to add materials to the database which do not come as standard with the model you can do it in the following way: Simply copy an existing material directory (say OghmaNano.local\oxieds\ito) to a new directory (say OghmaNano.local\oxieds\mynewmaterial). Then replace alpha.gmat and n.gmat with your data for the new material. You can ignore the data.json file, although if you know the energy levels you can add the values in the file.

If you don't have data to hand for your material, but you do have a paper containing the data, you use the program Engauge Digitizer, written by Mark Mitchell <https://github.com/markummitchell/engauge-digitizer> to export data from publications. After you have finished updating the new material directory, whenever a new simulation is generated the new material files will automatically be copied into the active simulation directory ready for use.

14.3 Adding new materials - the easy way

To add a new material go to the data base ribbon and click on *Materials database* as shown in figure 14.2.

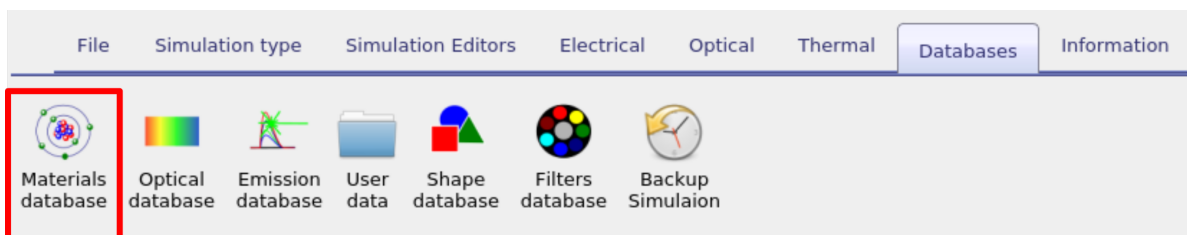


Figure 14.2: Opening the materials database

Then click *add material* in the top right of the window, this will bring up a dialogue box which will ask you to give a name for your new material, this is visible in figure 14.4.

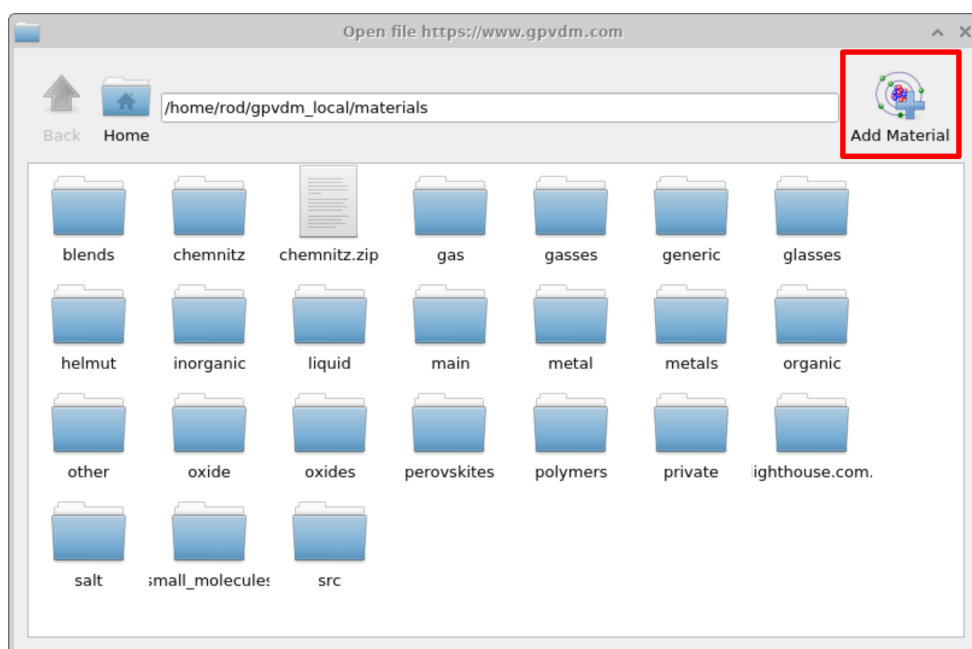


Figure 14.3: Select Add material

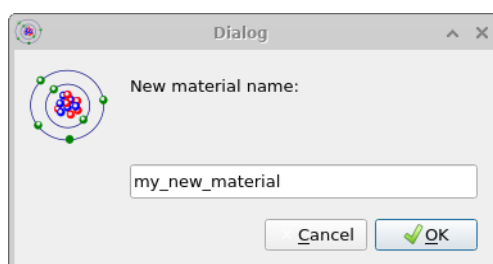


Figure 14.4: Type the name of the new material

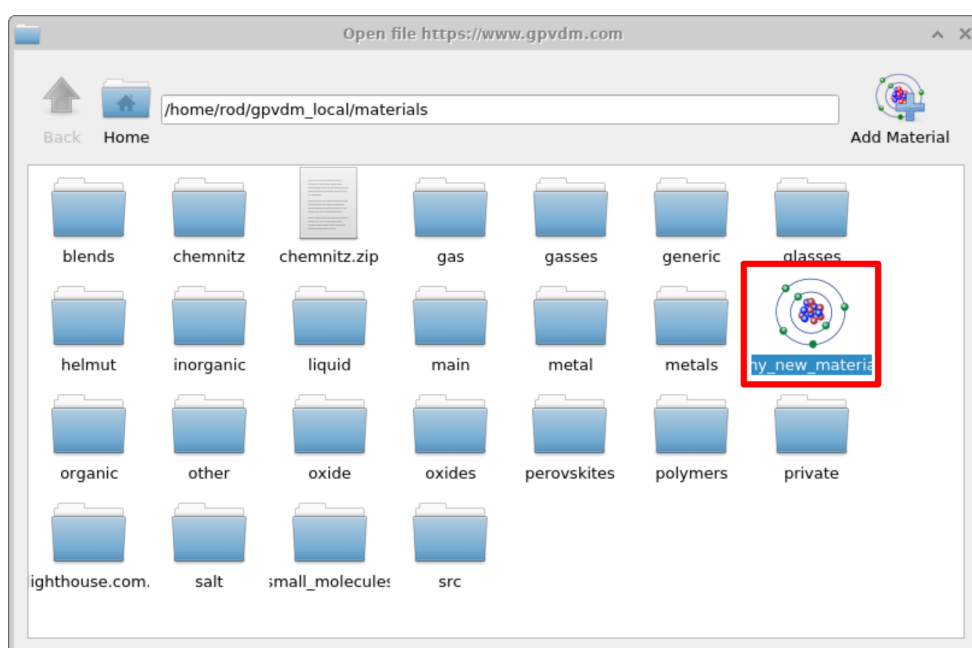


Figure 14.5: Open the new material

14.3. ADDING NEW MATERIALS - THE EASY WAY

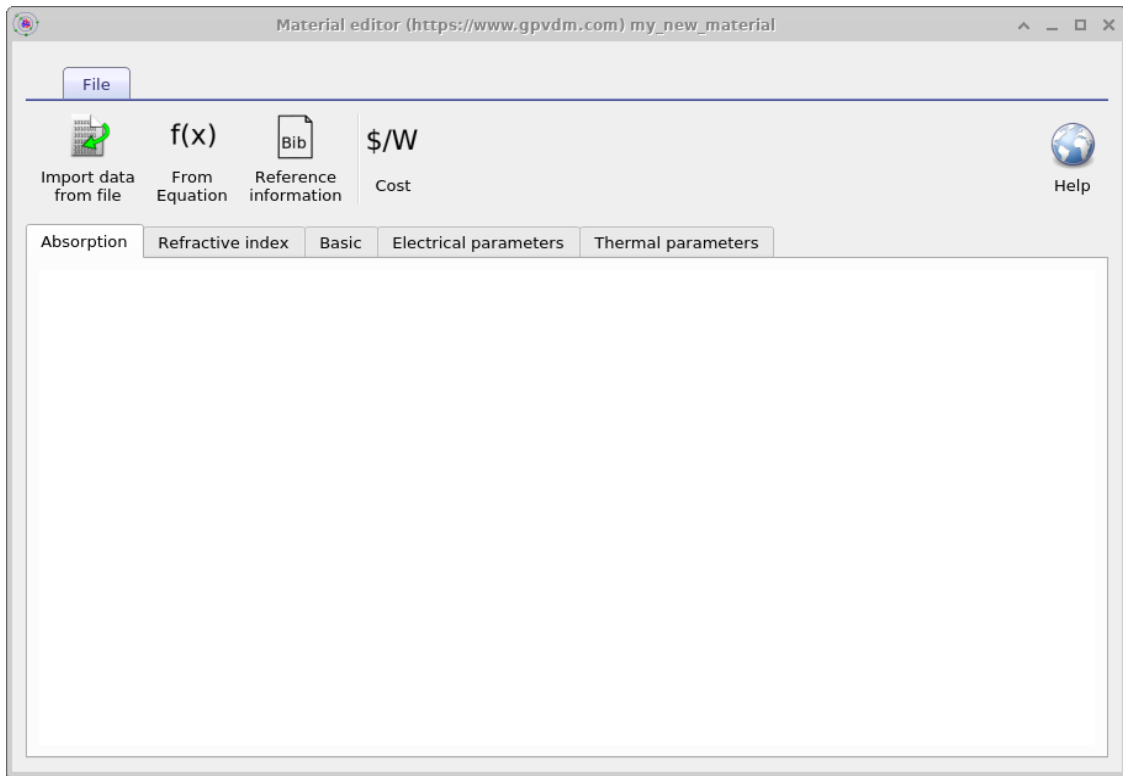


Figure 14.6: The new material without any data

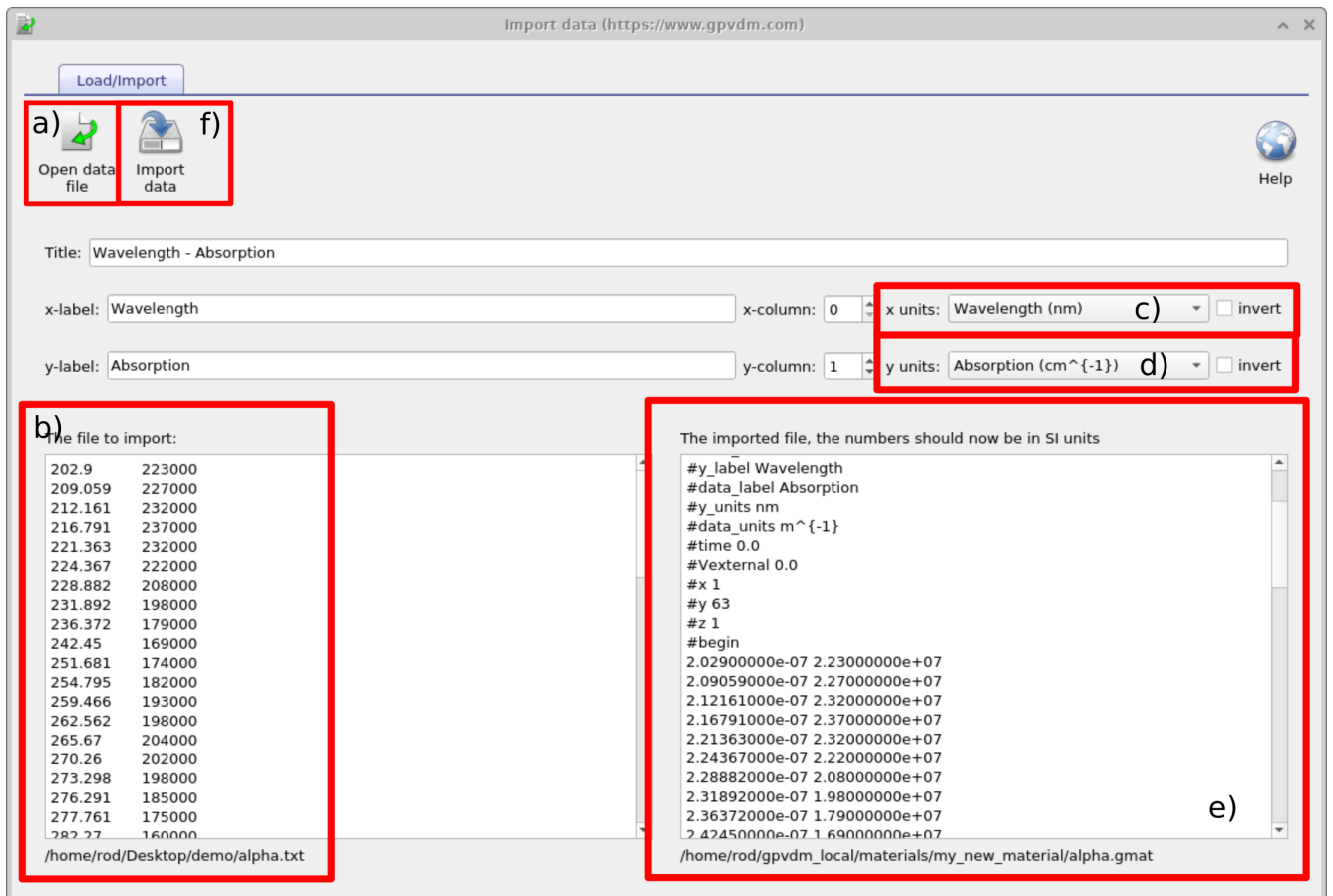


Figure 14.7: The data importer window

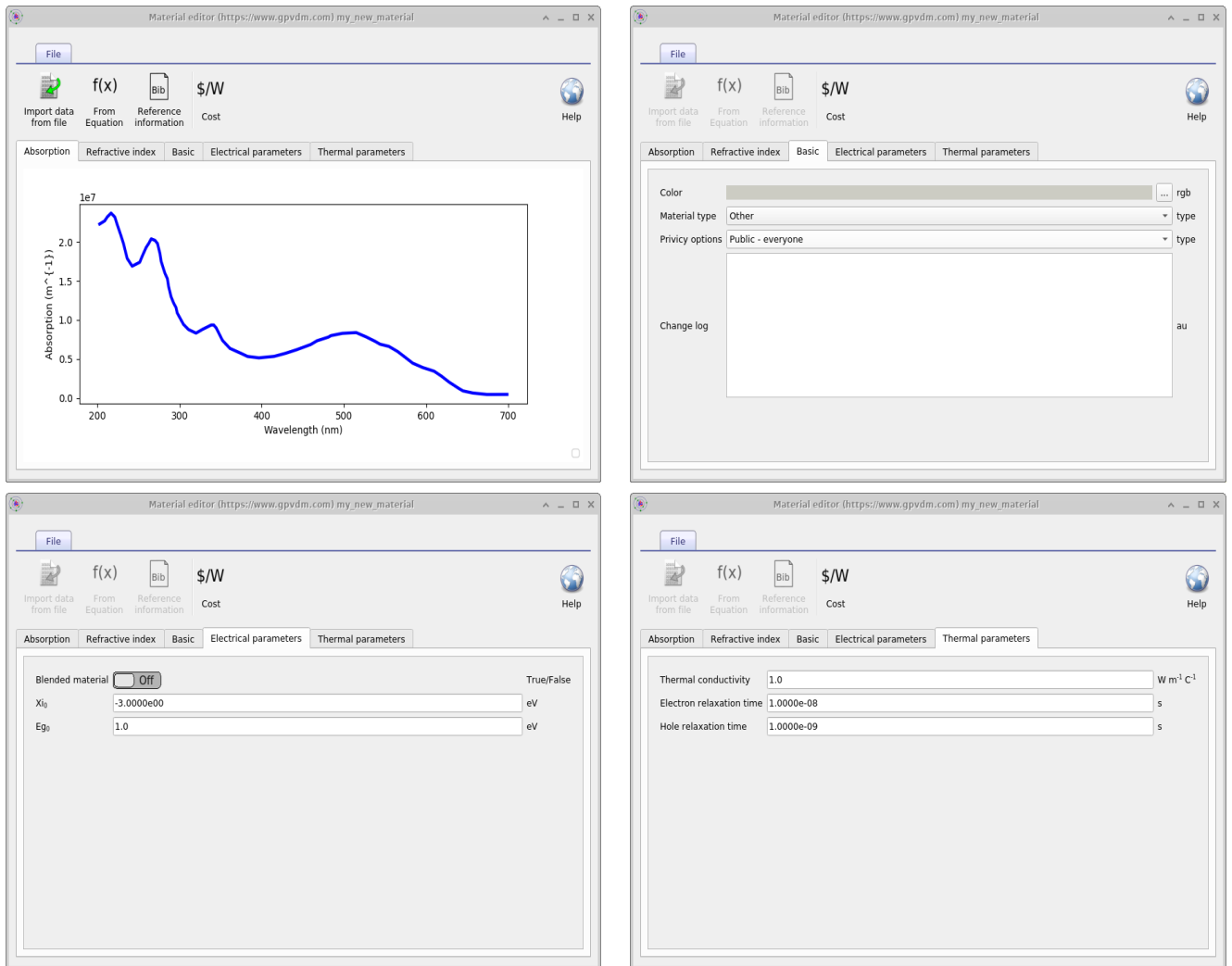


Figure 14.8: Clockwise from the top left; The imported absorption spectrum; The basic material parameters; The electrical parameters; and the Thermal parameters.

14.4 Emission database

This contains emission spectra for OLED materials.

14.5 Shape database

All physical objects within a simulation are *shapes*. For example the following things are all shapes; a layer of a solar cell; a layer of an OLED; a lens; a complex photonic crystal structure; contact stripe on an OFET; the complex hexagonal contact on a large area device (see figure 1.2 for more examples). These *shapes* are defined using triangular meshes for example a box which is used to define layers of solar cells, and layers of LEDs is defined using 12 triangles, two for each side. This box structure can be seen in figure 14.9.

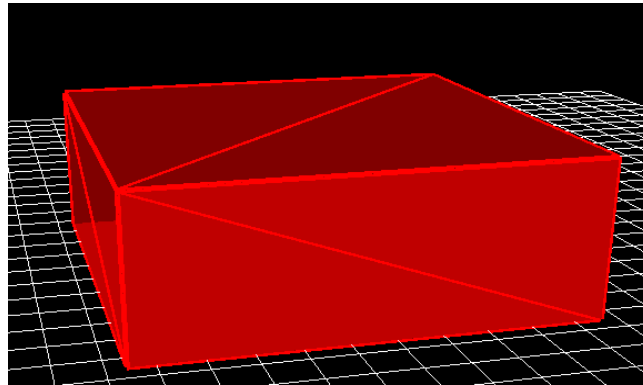


Figure 14.9: the box *shape*.

Shapes are stored in the shape database, this can be accessed via the database ribbon and clicking on the Shapes icon, see figure 14.10. By clicking on the *shape database* icon the shape database window can be brought up see figure 14.11.

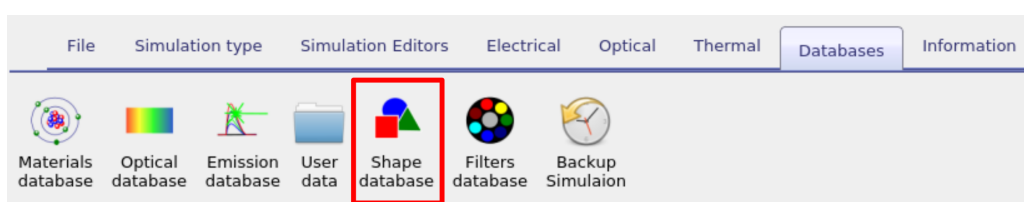


Figure 14.10: Opening the shape database

14.5. SHAPE DATABASE

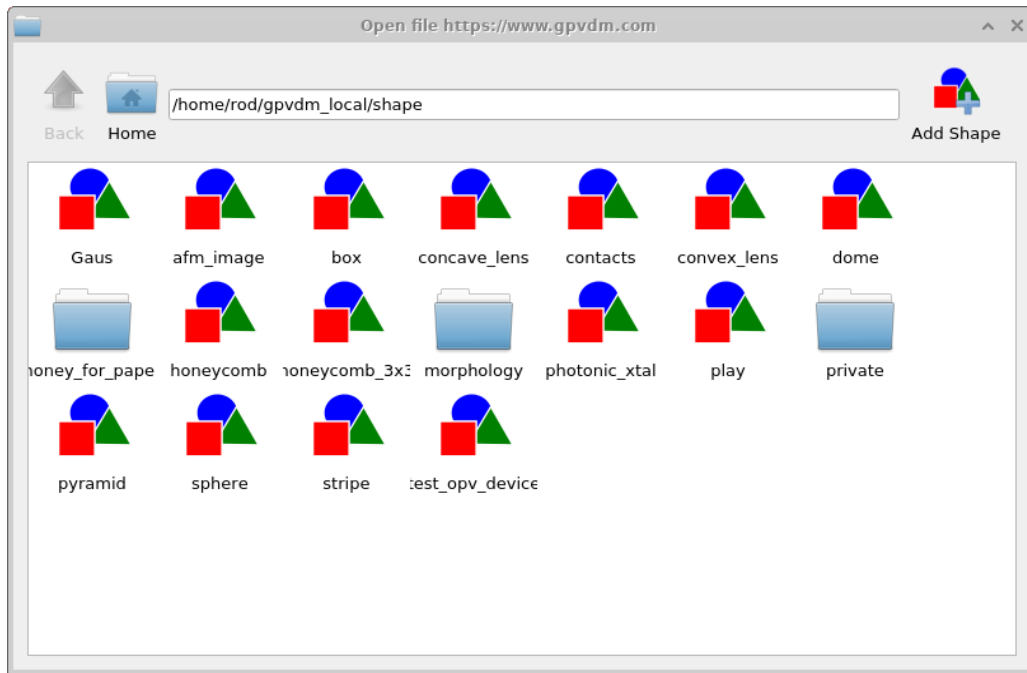


Figure 14.11: The shape database window

Try opening some of the shapes and have a look at them. You will get a window much like that shown in figure 14.12. Figure 14.12 shows a honeycomb contact structure of a solar cell. On the left of the window is the 3D shape, and on the right of the window is the 2D image which was used to generate it. Overlaid on the 2D image is a zx projection of the 3D mesh. The process of generating a shape involves first defining a 2D png image which you want to turn into a shape, in this case the 2D image is a series of hexagons and a bar at the top. This image is then converted into a triangular mesh using a discretization algorithm.

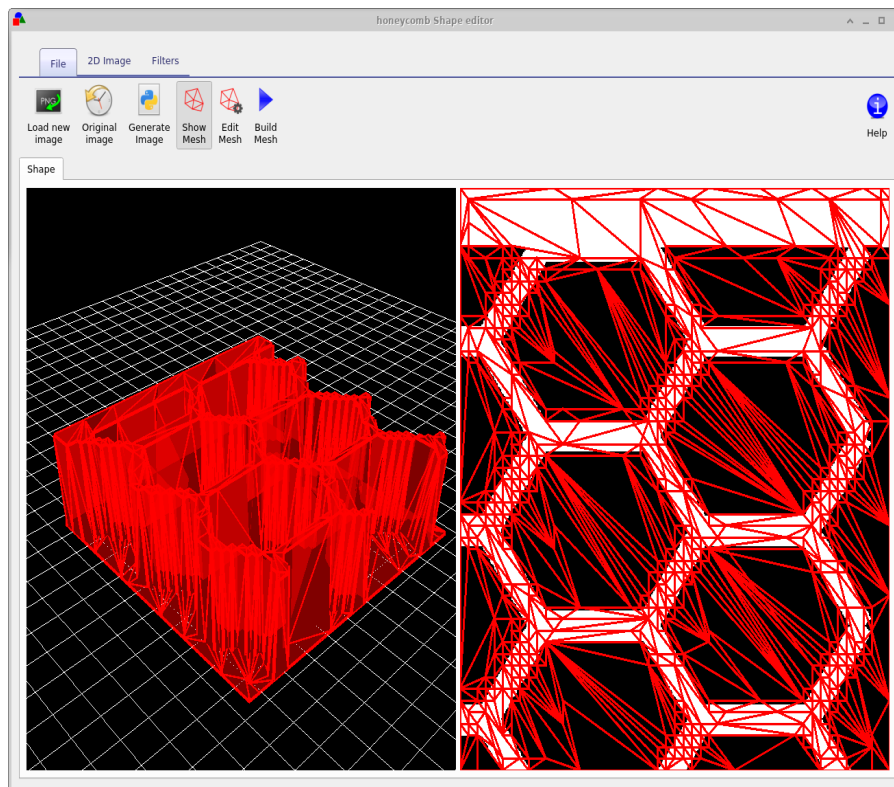


Figure 14.12: An example of a shape generated from a 2D png image. The 3D shape representing a hexagonal contact from a solar cell is on the left of the figure while the original 2D image is on the right.

Now try opening the shape *morphology/1* and you should see a window such as the one shown in figure 14.13, in the file ribbon find the icon which says *show mesh*. Try toggling it of and on, you will see the 2D mesh become hidden and then visible again. This example is a simulated bulk heterojunction morphology, but you can turn any 2D image into a shape by using the *load new image* button in the file ribbon. Try opening the mesh editor by clicking on *Edit Mesh* the *file* ribbon, you should get a window looking like figure 14.14. This configure window has three main options *x-triangles*, *y-triangles* and *method*. The values in *x-triangles*, *y-triangles* determine the maximum number of triangles used to discretize the image on the x and y axis. Try reducing the numbers to 40 then click on *Build mesh* in the file ribbon.

14.5. SHAPE DATABASE

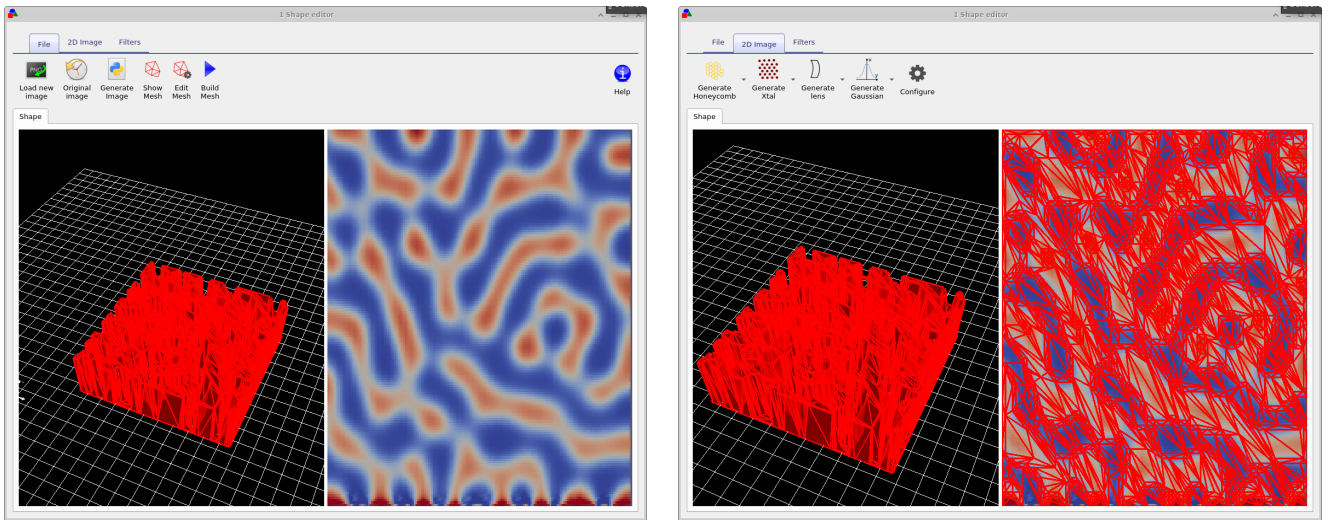


Figure 14.13: Clockwise from the top left; The imported absorption spectrum; The basic material parameters; The electrical parameters; and the Thermal parameters.

You will see that the number of triangles used to describe the image reduce. The more triangles that are used to describe the shape the more accurately the shape can be reproduced, however the more triangles are used the more memory a shape will take up and the slower simulations will run. There is always a trade off between number of triangles used to discretize a shape. Try going back to the *Edit Mesh* window and set *method* to *no reduce* and then click on *Build mesh* from the file menu again. You will see that the complex triangular mesh as been replaced by a periodic triangular mesh, which is more accurate but requires the full 70x70 triangles. The difference between the *no reduce* and *Node reduce* options are that *no reduce* simply uses a regular mesh to describe and object and *Node reduce* starts off with a regular mesh then uses a node reduction algorithm to minimize the number of triangles used in the mesh.

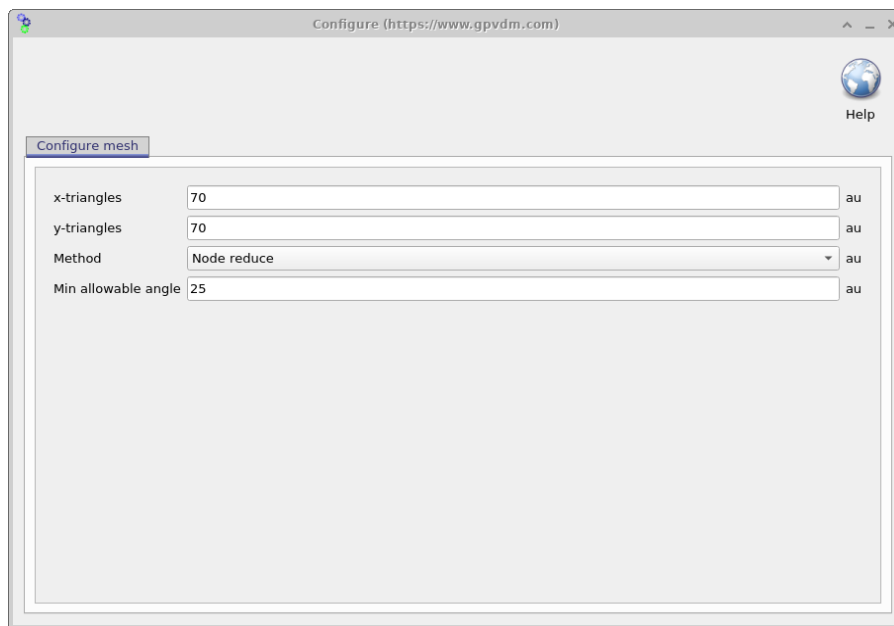


Figure 14.14: The mesh editor window, accessed via the file ribbon.

As well as loading images from file, the shape editor can generate it's own images for standard objects used in science, the 2D image ribbon is visible in the right hand panel of figure 14.13. There are options to generate lenses, honeycomb structures and photonic crystals.

Each button has a drop down menu to the right of it which can be used to configure exactly what shape is generated.

The final ribbon to be discussed is the *Filters* ribbon. This is used to change loaded images, try turning on and off the threshold function. This applies a threshold to an image so that RGB values above a given value are set to white and those below are set to black. There are also other functions such as Blur, and Boundary which can be used to blur and image and apply boundaries to an image.

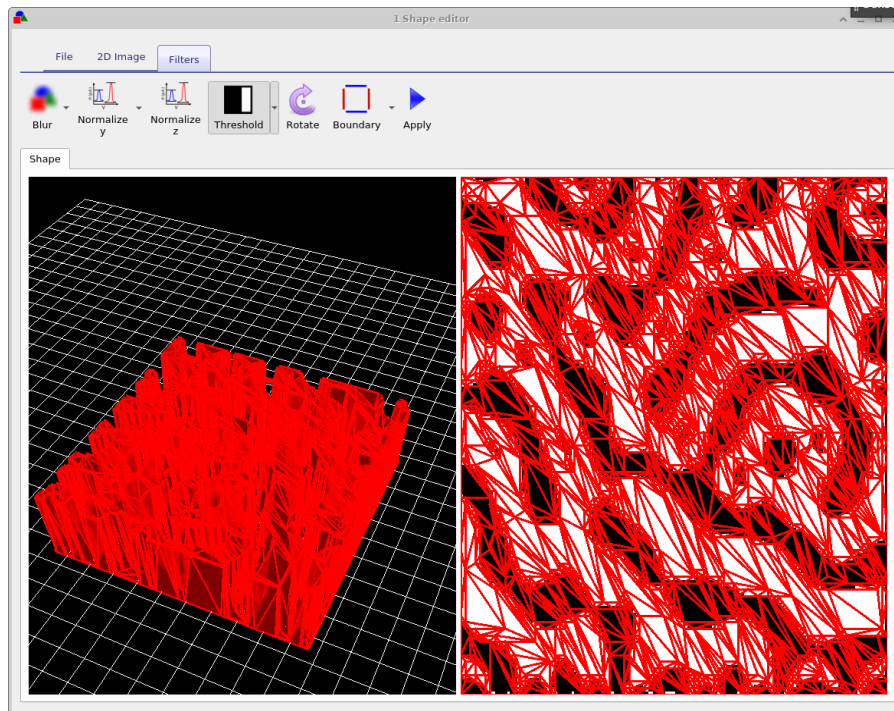


Figure 14.15: The shape database

14.5.1 The shape file format

A shape has to be a fully enclosed volume, if you use the built in shape discretizer this will be done for you automatically. However if you are building shapes by hand you will have to enforce this condition. Each shape directory contains the following files

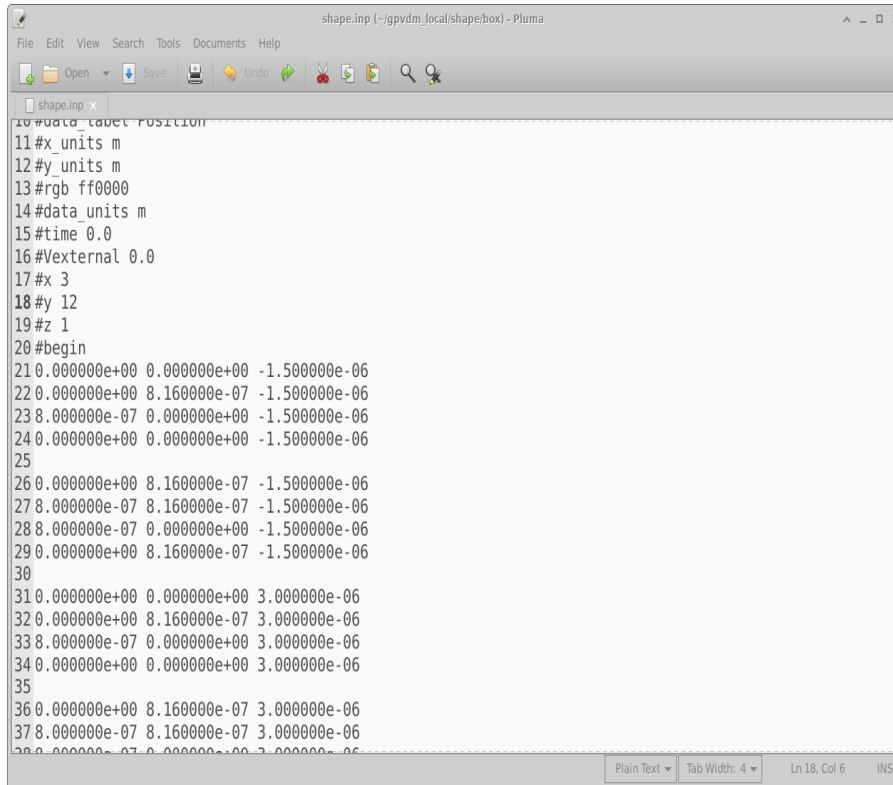
File name	Description
<i>data.json</i>	Holds the configuration for the shape file
<i>image_original.png</i>	backup of the imported image
<i>image_out.png</i>	The final processed image
<i>image.png</i>	The imported image which may be modified.
<i>shape.inp</i>	The discretized 3D structure.

Table 14.1: The files within a shape directory

The png files are of images in various states of modification. The data.json file stores the configuration of the shape editor and the shape.inp file contains the 3D structure of the object. An example of a shape.inp file is shown below in 14.16. The file format has been written so that gnuplot can open it using the splot command without any modification. As such each triangle is comprised of four z,x,y points (lines 21-24), the first three lines define the triangle, and the fourth line is a repeat of the first line so that gnuplot can plot the triangle nicely. The number

14.5. SHAPE DATABASE

of triangles in the file is defined on line 18 using the `#y` command. The exact magnitudes of the `z,x,y` values do not matter because as soon as the shape is loaded all values are normalized so that the minimum point of the shape sits at 0,0,0 and the maximum point from the origin sits at 1,1,1. When being inserted into a scene, the shape is then again renormalized to the desired size of the object in the device.



```
shape.inp (-/gpvdm_local/shape/box) - Pluma
File Edit View Search Tools Documents Help
Open Save Undo
shape.inp x
10 #data_label POSITION
11 #x_units m
12 #y_units m
13 #rgb ff0000
14 #data_units m
15 #time 0.0
16 #Vexternal 0.0
17 #x 3
18 #y 12
19 #z 1
20 #begin
21 0.000000e+00 0.000000e+00 -1.500000e-06
22 0.000000e+00 8.160000e-07 -1.500000e-06
23 8.000000e-07 0.000000e+00 -1.500000e-06
24 0.000000e+00 0.000000e+00 -1.500000e-06
25
26 0.000000e+00 8.160000e-07 -1.500000e-06
27 8.000000e-07 8.160000e-07 -1.500000e-06
28 8.000000e-07 0.000000e+00 -1.500000e-06
29 0.000000e+00 8.160000e-07 -1.500000e-06
30
31 0.000000e+00 0.000000e+00 3.000000e-06
32 0.000000e+00 8.160000e-07 3.000000e-06
33 8.000000e-07 0.000000e+00 3.000000e-06
34 0.000000e+00 0.000000e+00 3.000000e-06
35
36 0.000000e+00 8.160000e-07 3.000000e-06
37 8.000000e-07 8.160000e-07 3.000000e-06
38 0.000000e-07 0.000000e+00 3.000000e-06
Plain Text Tab Width: 4 Ln 18, Col 6 INS
```

Figure 14.16: An example of the `shape.inp` file.

14.6 Filters database

This contains optical filters.

14.7 Backups of simulations

Very often when running a simulation you want to make a copy of it before continuing to play with the parameters. To do this click on the backup simulation button in the database ribbon (see figure 14.1), this will bring up the backup window, see figure 14.17. If you click on the "New backup" icon on the top right of the window, a backup will be made of your current simulation. And an icon representing the backup will appear in the backup window. To restore the backup double click on the icon representing your stored simulation. Note this backup is only stored in the your local simulation directory, and is more of a checkpoint than a real backup.... so make sure you have other copies of your simulation if it is very important to you..

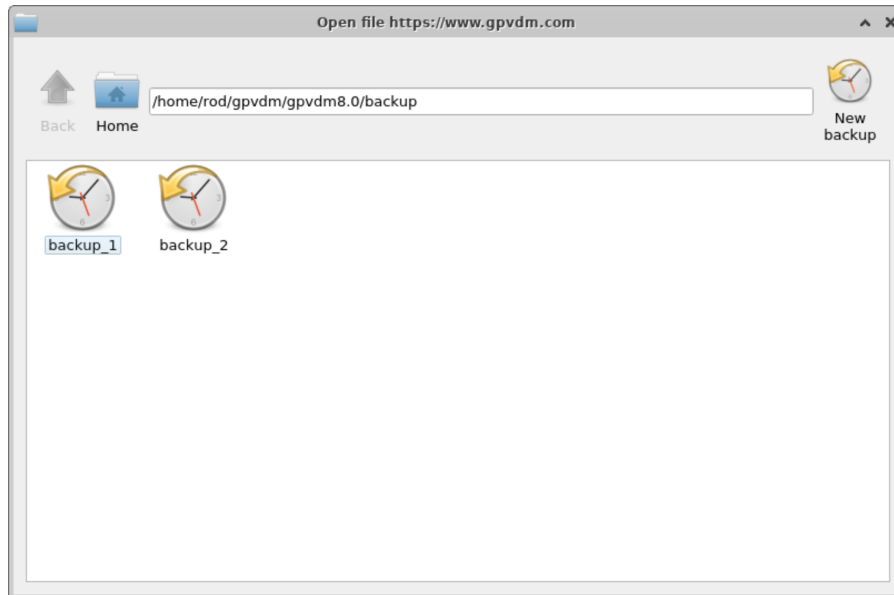


Figure 14.17: Backing up a simulation

Chapter 15

Fitting experimental data

Related YouTube videos:



Advanced topics in fitting of JV curves to experimental data using OghmaNano.



Fitting transient photocurrent (TPC) and light JV curves using OghmaNano



Fitting the light JV curve of an ultra large area (2.5meter x 1cm) OPV device using OghmaNano

In the same way you can fit the diode equation to a dark curve of a solar cell to extract the ideality factor, OghmaNano can be fitted to experimental data using the fitting function. Fitting is a good way to extract physical parameters from a device such as mobility, density of trap states etc. The advantage of fitting a complex model such as OghmaNano to data rather than simplistic analytical equations is that far more detailed information can be extracted and a better physical picture of the underlying physics obtained. This section gives an overview of the fitting tools in OghmaNano.

15.1 Key tips and tricks

- Generally speaking fitting is a tricky process requiring a lot of patience and manual fine tuning to get it to work. Don't expect to click a button and for it to just work. You will have to work to get nice fits.
- If the fit is not working something may be wrong with the physical assumptions you have made with your device. The model will only fit physically reasonable data so if something is off by an order of magnitude go back and think again about what you are asking the model to do.
- Different data sets provide different types of information. For example the dark JV curve of a solar cell provides information about the shunt resistance, the series resistance, and some information about mobility and recombination/trap states. However, the light JV curve provides almost no information about the shunt resistance so don't expect a fit to the light JV curve to provide accurate estimates of R_{shunt} . Always think about what information is contained within your data before interpreting the numbers extracted from a fit.
- The fitting works process by: 1) Running a simulation; 2) Calculating the difference between the numerical and experimental results; 3) tweaking the simulation parameters

15.2. THE MAIN FITTING WINDOW

; 4) rerunning the simulation and seeing if the difference between the experiment and simulation has reduced; 5) If the error has reduced the change of parameter is accepted and the process repeated with a different parameter. This process continues hundreds or thousands of times until an acceptable fit has been achieved. Therefore to do a fit the model must be run thousands of times, this means that for a fit to arrive at an answer quickly, the individual simulation when run alone must be fast. So for example if your simulation has 1000 mesh points, when fitting try reducing this to 10. Or if you have 1000 time steps try reducing this to 100. Every speed up in the base simulation will result in a speed up in the fitting process.

- Writing files to disk is the slowest part of any computational process. Even modern SSDs are about 30 times slower than main memory for example the maximum write speed to an SSD is 456 MB/s where as the bandwidth of a PC3-12800 memory module is 12,800 MB/s. When you save your files on USB drives, network storage drives, or even worse the internet aka OneDrive or Dropbox, read write speeds again drop massively. Therefore if you want your simulation to run fast save it on a local hard disk which is ideally and SSD and not a mechanical drive and not being mirrored over the network.
- As stated above writing files to disk is slow, therefore try to minimize the number of files your simulation kicks out. Turn off things such as snapshots, optical output and the dynamic folder. You can check if your simulation is dumping a lot of files by opening your simulation directory in windows explorer and counting the files. A simulation set to write very little to disk should have about 50 files in it. If your simulation directory has hundreds of files in it then you need to find out why.
- Although you can fit with the GUI, it can be slow. I personally tend to set up fits in the GUI but run them from the command line. There is a section on how to do this below.
- Fitting writes quite a lot of files to disk. Virus killers can slow down the fitting significantly as they scan all the data files before they are written to disk.

15.2 The main fitting window

An example of how to fit the model to experimental data is included in the demo simulations provided with OghmaNano. This can be found under "Fitting and parameter extraction" (see ??). If you open this simulation you will be presented with

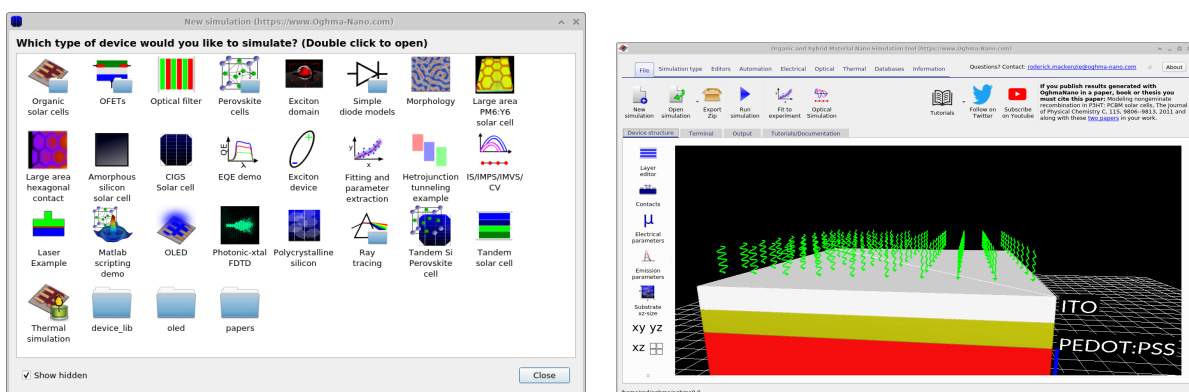


Figure 15.1: a) The example fitting simulation can be found in the "Fitting and parameter extraction" demo. b) Once opened it will look like the window on the right.

From the file ribbon then click on the "Fit to experiment icon". This will bring up the fitting window (see figure 15.2). The icons in the ribbon of this window perform the following tasks:

- New experiment: Currently the window is only displaying one set of experimental data in this case a light JV curve. Using the New experiment button one can add other data sets such as a dark JV curve to the fit. The more sets of data you fit against the more accurate your extracted parameters will be but the harder the fit will be to perform and the slower it will run.
- Delete experiment: This will remove a data set from the fit.
- Clone experiment: This will clone the current data set to a new data set.
- Rename experiment: This will rename the data set.
- Export data: This will export the fit to a zip file.
- Import data: This will import experimental data. The import wizard is explained elsewhere.
- Configure: This is used to configure the fitting variables, this is explained in detail below.
- One iteration data: This will run the fit just once to see how close to the experimental data it is. It is highly recommended to use this function and change the parameters by hand to get a closish fit before running the automated fit.
- Run fit: This will run the automated fitting algorithm. It will run forever, you have to press the button again to stop it.
- Fit this data set: This enables or disables the fitting of the data set currently being viewed.

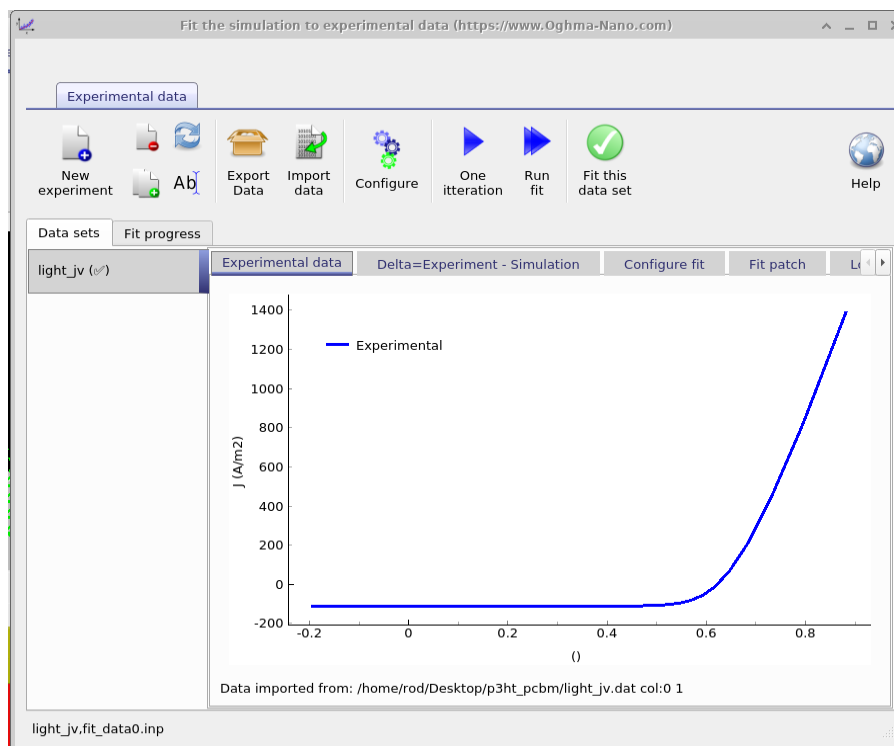


Figure 15.2: The main fitting window.

15.3. FIT VARIABLES

If you click the one fit button, the fit window will update and will look like ??a. You can now see the difference between the experimental and simulated curves. The green line represents the difference between the two curves, it is called the error function. It represents the mathematical difference between the simulated and experimental data. If you now click "Run fit" to start the fitting process, you should see the curves gradually start to get closer and the error function decrease in value. Now click on the "Fit progress" tab, this plots the error function as a function of fit iterations. Watch as the error drops off. It should start looking like figure ??b. This process should take about 30 seconds, if it takes longer read section 15.1 above.

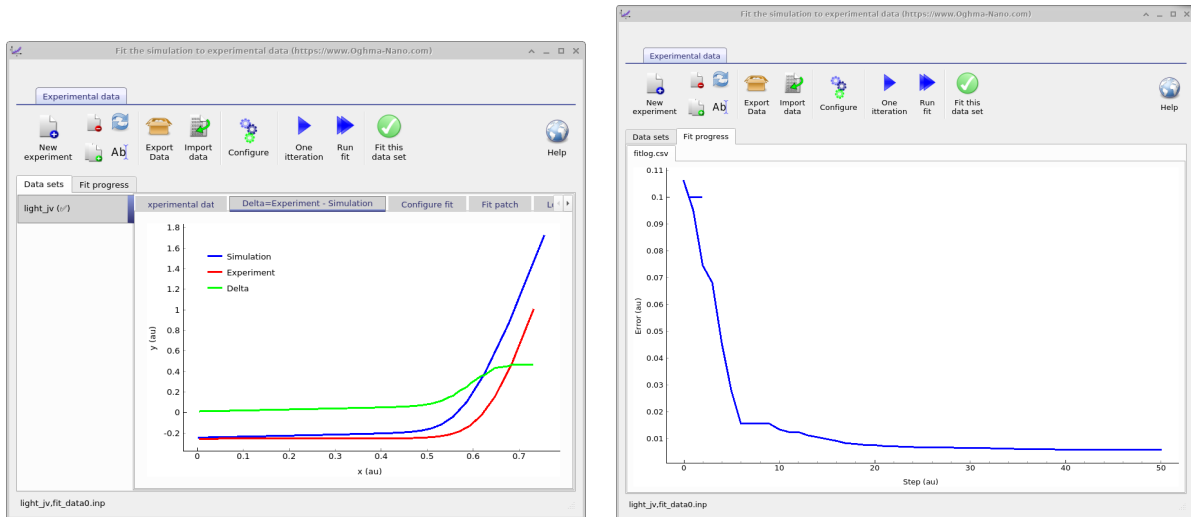


Figure 15.3: a) The result of clicking the one fit button. b) The error function dropping during a simulation.

15.3 Fit variables

In figure 15.2, there is a "Configure" icon. If you click on this it will open the fit variable window. This window is used to configure the fitting variables. This window has the following tabs:

- Duplicate variables: This is used to copy one parameter to another. In this example we are assuming the device is symmetric, so we are only fitting the electron but we are using the "Duplicate variables" variables tool to copy the newly fitted electron parameters on top of the hole parameters. This enables us to vary electron mobility in the fitting process but let the hole mobility have the same value at each step. (See figure ??).
- Fit variables: This defines the variables we will fit. The more variables you fit at the same time the longer the fit will take. Try to minimize the number of variables you are fitting. A good tip is to start of fitting with symmetric parameters then only move to asymmetric parameters, once the fit becomes good. (see figure 15.5).
- Fit variables: This is used to force one parameter to be bigger than another.
- Configure minimizer: This configures how the fitting algorithm behaves.
 - Stall steps: If fit error does not improve in this number of steps then the fit is considered stalled and therefore stopped.

- Disable reset at level: This will stop the fit restarting if fit error drops to below this level.
- Fit define convergence: This is the level of error at which the fit will stop.
- Start simplex step multiplication: This defines the size of the first fitting step, a smaller value will mean the fitting algorithm will only change the initial numbers a bit, while a large number will change the initial numbers a lot. If you want your fit to explore the parameter space widely set this value to be greater than 1.0. If you want your fit to more or less stay around where your initial parameters are set this value to be less than 1.0. A value of 2.0 is considered big, a value of 0.1 is considered small.
- Fitting method: This can be set to Simplex (simplex downhill https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method) or newton. Newton is experimental and should not be used.
- Enable snapshots during fit: By default snapshots are turned off during fitting as they produce a lot of disk access to allow them to be dumped to disk set this on.
- Simplex reset steps: This sets after how many steps the simplex algorithm is reset. Resetting the algorithm can push the answer away from the solution, but it can also pop the solver out of a valley if it has become trapped and allow better convergence.

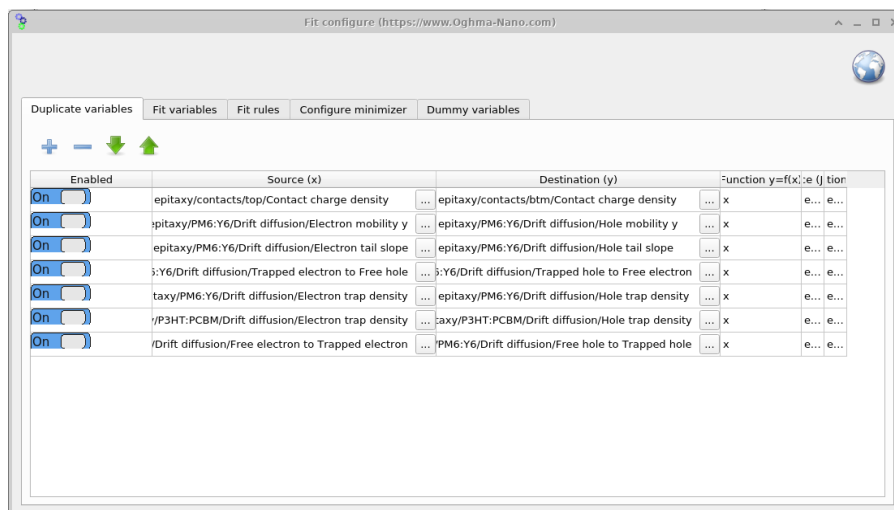


Figure 15.4: The fit variable window. This window is used to configure the fitting variables.

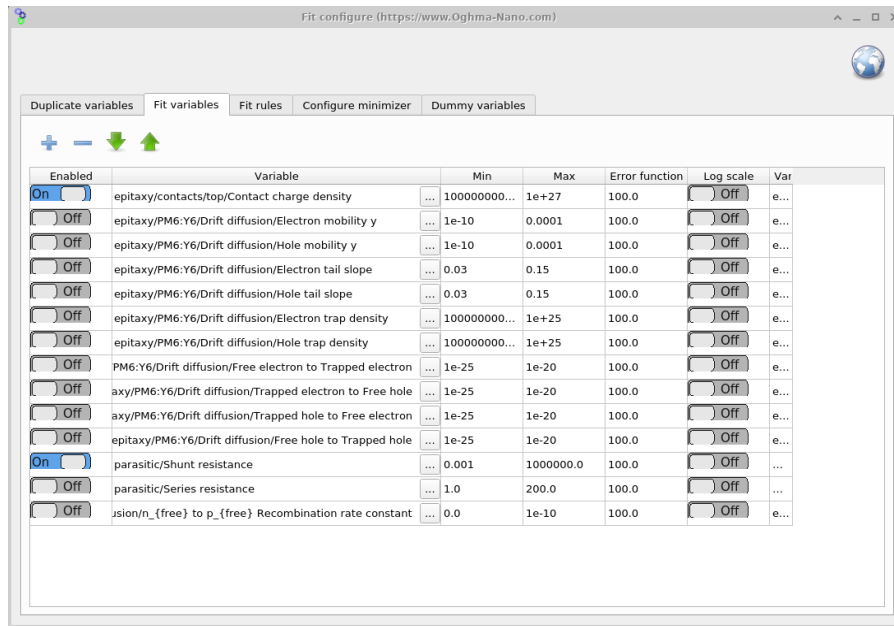


Figure 15.5: The fit variables used to fit the data.

15.4 How the fitting process works

When you click the "Run fit" button, OghmaNano makes a new directory inside the simulation directory called "sim" this is the directory in which the fitting process takes place. Inside this directory OghmaNano will make one new directory for each data set you are trying to fit, it will populate each directory with the sim.json (and sim.oghma) files from your main simulation directory. At this point the sim.json files in all the directories are identical. Then using the contents of the fit "fit patch" (see figure 15.6) the content of each sim.json file will be updated, this process is called patching the simulation files. This process enables you to adjust parameters in each simulation directory to match the data set you are trying to fit. For example you might want one data set to have optical/light/Psun set 1.0 and another to be set to 0.0 to enable fitting of a 1 sun JV curve and a dark JV curve. After patching each directory, the fitting process then commences. During this process fitting variables in the sim.json files in the "sim" directory are updated. During the fit the algorithm will often produce fits which are worse than the current best effort, and only sometimes produce fits which are better than the current best effort. Only when a better fit is obtained will the sim.json file be updated in the main simulation directory and the curves in the GUI also updated.

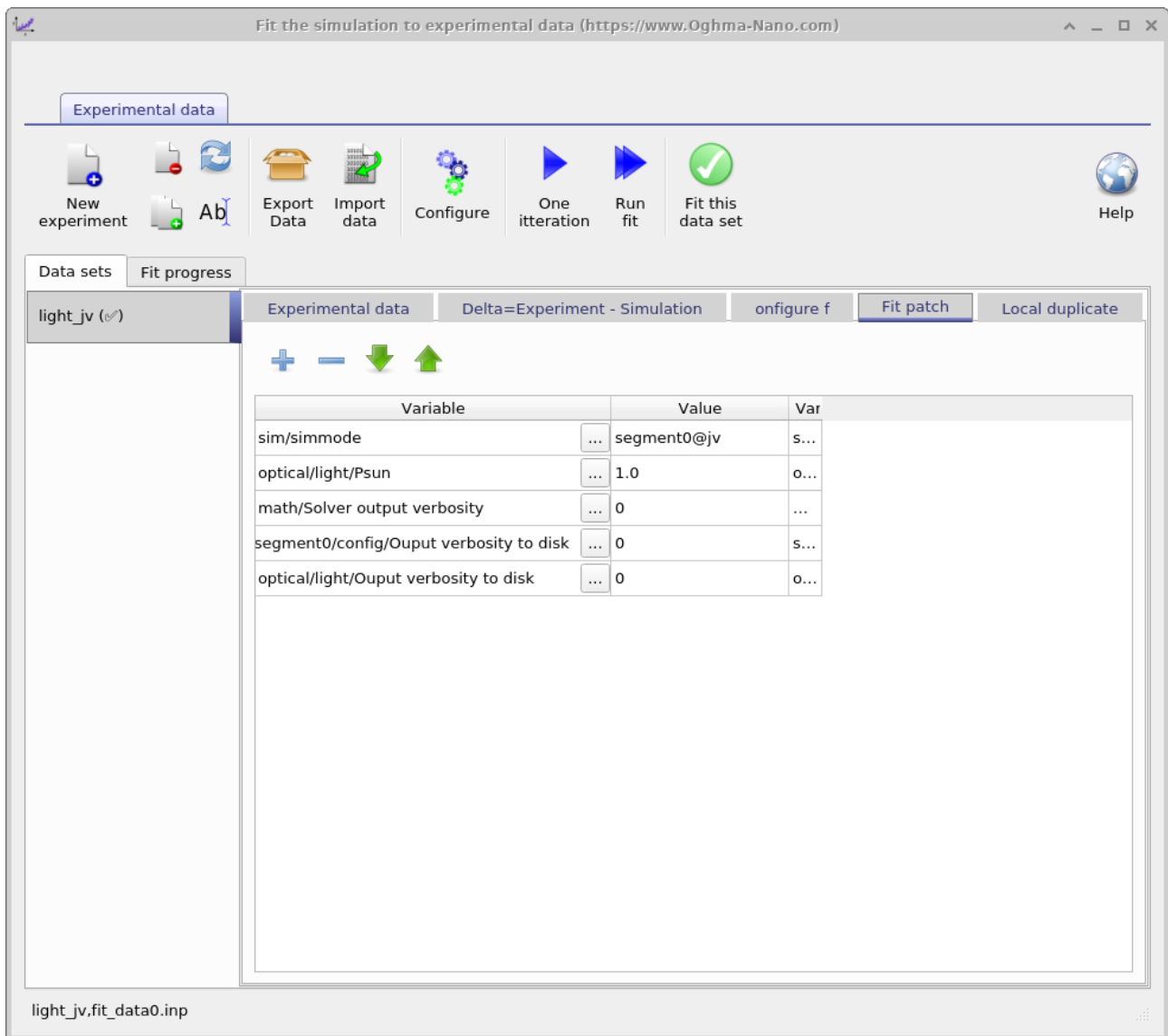


Figure 15.6: The fit patch applied to each data set.

15.5 Fitting without the GUI

The GUI is a very easy and efficient way to setup a fit. However, it takes considerable CPU time to update the user interface as the fit runs and this therefore slows the fitting process. Therefore if you are doing lots of fitting or fitting difficult problems, fitting without the GUI can be faster. This section covers how to fit from the Windows command line:

1. First set up your simulation you want to fit in the usual way using the GUI. Run a single iteration of the fit to make sure it looks right. Then close the GUI.
2. Next we need to tell Windows where it can find OghmaNano, usually it has been installed in `C:\Program files x86 \OghmaNano`. If you open this directory you will see lots of files. But the two key ones are `oghma.exe` and `oghma_core.exe`. The file `oghma.exe` is the GUI, `oghma_core.exe` is the core solver, these are completely independent programs. The core solver can be run without the GUI. To tell windows where these files are we need to add `C:\Program files x86 \OghmaNano` to the windows path. This can be done by following these <https://docs.microsoft.com/en-us/previous-versions/>

15.5. FITTING WITHOUT THE GUI

`office/developer/sharepoint-2010/ee537574(v=office.14)` instructions. These instructions are for a modern version of Windows, but on your system things may be in slightly different places. On most versions of windows the process is more or less the same, if you get stuck google "adding a path to window".

3. Click on the start menu and type "cmd" and enter to bring up a Windows terminal. Type:

```
oghma_core.exe --help
```

Note it is a double dash before help not a single dash.

This should bring up some help for OghmaNano. If it does then we have successfully told windows where `oghma_core.exe` lives. If you get an error, try step 2 again (and/or restart your computer).

4. Now that windows knows where `oghma_core.exe` lives, we can navigate to our simulation directory. Use `cd` to navigate to the directory where your simulation you want to fit is saved.
5. First run the command `oghma_core.exe` to see if your simulation runs OK. If it does not then recheck your simulation file.
6. Now run a single fit by typing:

```
oghma_core.exe --1fit
```

Inspect the results in the "sim" directory, use your favourite plotting program to compare the results to the experimental data. Note the experimental data is stored in `fit_data(0-1).inp`.

7. If everything went well with the above step, you can run a real fit by typing:

```
oghma_core.exe --fit
```

Again those are double dashes before the fit command. Ctrl+C will terminate the fit. You can check the progress of convergence by plotting `fitlog.csv`.

Chapter 16

Automation and Scripting

Often a user will have set up a simulation structure to represent a real world device but will then ask the question: What happens to my solar cell efficiency as I change the mobility of the active layer? Or what happens to the wavelength of my laser output as I change the thickness of the Quantum Well. To answer these type of questions one must change one or more material parameters over a range of values and then examine the simulation results. Clearly this could be done by hand but there are better ways to automate this process.

There are three main ways to automate OghmaNano simulations:

1. The first method is using the parameter scan window, this is described below in section 16.1. The parameter scan window allows a user to vary a parameter (or multiple parameters) in steps using the graphical user interface. No knowledge of coding is required for this approach. The parameter scan window is useful if one wants to quickly examine how a parameter influences the results and the scenario you are examining is not very complex. The scan window fits most users's needs most of the time.
2. For more fine grained control over how the parameters are varied the next method is to use Python scripting, this is described in section 16.3.1. Python scripting allows the user ultimate flexibility in adjusting all simulation parameters and running simulations, Python is widely available which makes this approach very attractive.
3. The third way is through MATLAB scripting, this is described in section 16.3.2. The advantage of MATLAB scripting is that lots of people can code in MATLAB so makes automating OghmaNano very accessible. The downside of using MATLAB is that it is quite expensive and not all people have access to it. An alternative to MATLAB would be Octave however at the time of writing it does not have a json reader/writer.

Or option 4: All the above methods rely on the same principles: The OghmaNano simulation save file is systematically edited and the back end of the software *oghma_core.exe* run on the *sim.oghma* file to generate new results. Key to understanding how scripting works is to realize that the *sim.oghma* is simply a zip file (See 13.1) with a json file (*sim.json*) inside it, and if one can edit the json file (using any language you want ActionScript, C, C++, C#, Cold Fusion, Java, Lisp, Perl, Objective-C, OCAML, PHP, Python, Ruby etc...) then you can automate OghmaNano.

16.1 The parameter scan window

Related YouTube videos:



Using the parameter scan tool in OghmaNano

The most straight forward way to systematically vary a simulation parameter is to use the scan window. In this example we are going to systematic change the mobility of the active layer of a PM6:Y6 solar cell, you can find this example in the example simulations under *Scripting and fitting/Scan demo (PMY:Y6 OPV)*. Once you have located this simulation and opened it, you then need to bring up the parameter scan window, this can be done by clicking on the *Parameter scan* icon in the Automation ribbon (see Figure 16.1). Then make a new scan by clicking on the *new scan* button (1) (In the example simulation this has already been done for you). Open the new scan by double clicking on the icon representing the scan (2), see figure 16.2. This will bring up the scan window, see figure 16.3.

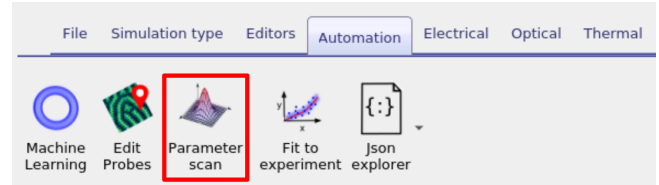


Figure 16.1: Step 1: Select the Parameter scan tool, to bring up the parameter scan window.

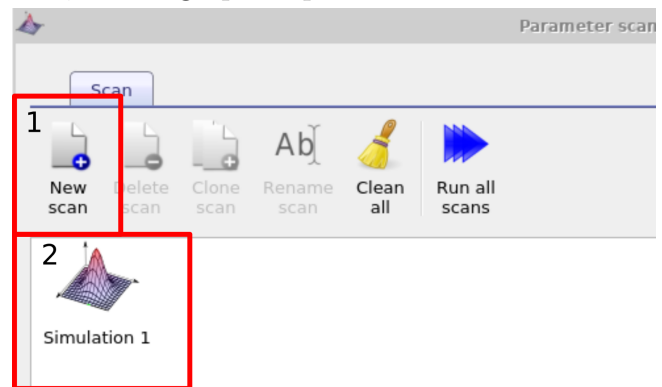


Figure 16.2: Step 2: Make a new parameter scan, then double click on it to open it.

16.1.1 Changing one material parameter

Once the *scan window* has opened, make a new scan line by clicking on the the plus icon (1) in figure 16.3, then select this line so that it is highlighted (2), then click on the three dots (3) to select which parameter you want to scan. Again if you are using the example simulation this will already have been done for you.

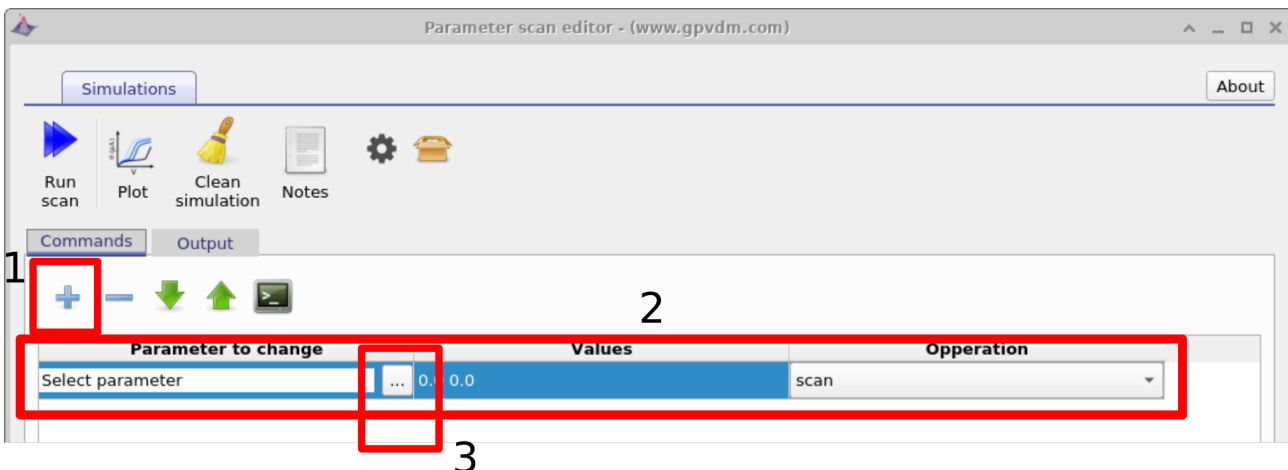


Figure 16.3: Step 3: Add a 'scan line' to the scan.

In this example we will be selecting the electron mobility of a PM6:Y6 solar cell. Do this by navigating to epitaxy→ PM6:Y6→ Drift diffusion→ Electron mobility y. Highlight the parameter and then click OK. This should then appear in the scan line. The meaning of *epitaxy→ PM6:Y6→ Drift diffusion→ Electron mobility y* will now be explained below:

- epitaxy: All parameters in the .oghma file are exposed via the parameter selection window see 16.4. This file is a tree structure, see 13.1. The device structure is defined under the heading epitaxy.
- PM6:Y6: Under epitaxy each layer of the device is given by its name. The active layer in this device is called PM6:Y6, if your active layer was called Perovskite or P3HT:PCBM you would have selected this instead.
- Drift diffusion: All electrical parameters are stored under the sub heading *drift diffusion*.
- Electron mobility y: One can define asymmetric mobilities in the z,x and y direction - this is useful for OFET simulations. However by default the model assumes a symmetric mobility which is the same in all directions. This value is defined by *Electron mobility y*.

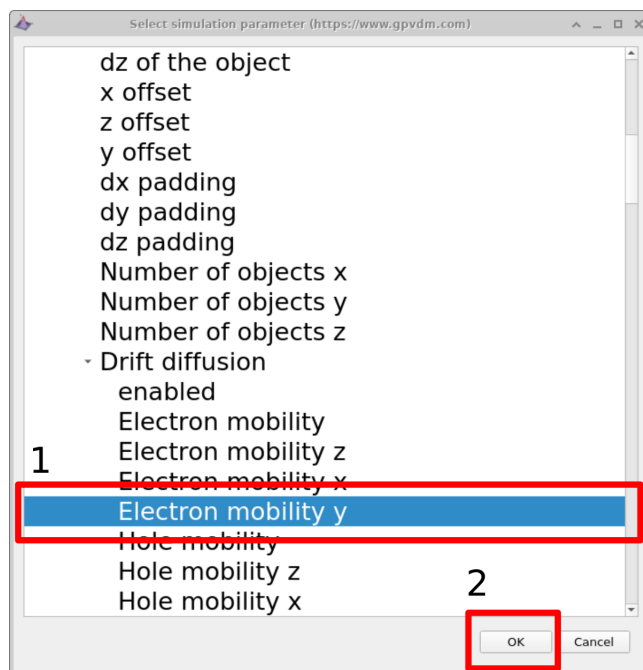


Figure 16.4: Step 5: Select the parameter you want to scan in the parameter selection window, in this case we are selecting epitaxy→ PM6:Y6→ Drift diffusion→ Electron mobility y.

Next enter the values of mobility which you want to scan over in this case we will be entering $1e-5$ $1e-6$ $1e-7$ $1e-8$ $1e-9$ (see figure 16.5 1) then click *run scan* (see figure 16.5 2). OghmaNano will run one simulation on each core of your computer until all the simulations are finished.

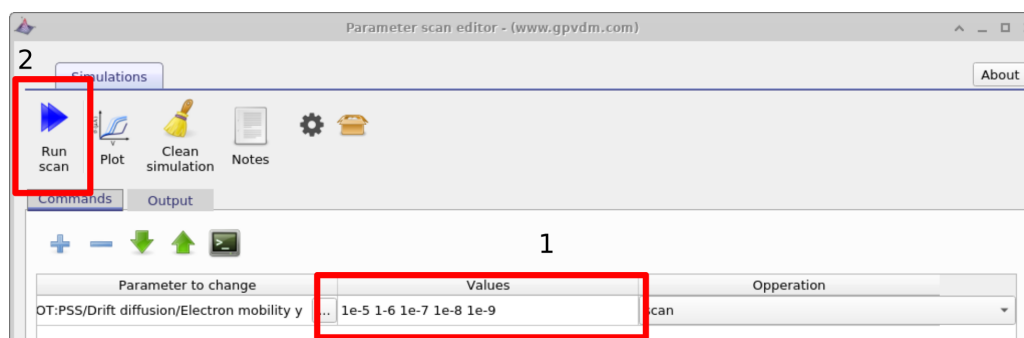


Figure 16.5: Step 6: Enter the input values of mobility (or other values) you want to scan over (1). Then run the simulations.

To view the simulation results click on the *output* tab this will bring up the simulation outputs, see figure 16.6. You can see that a directory has been created for each variable that we scanned over so $1e-5$, $1e-6$, $1e-7$, $1e-8$ and $1e-9$. If you look inside each directory it will be an exact copy of the base simulation directory. If you double click on the files with multi-colored JV curves, see the red box in figure 16.6. OghmaNano will automaticity plot all the curves from each simulation in one graph, see figure 16.7.

16.1. THE PARAMETER SCAN WINDOW

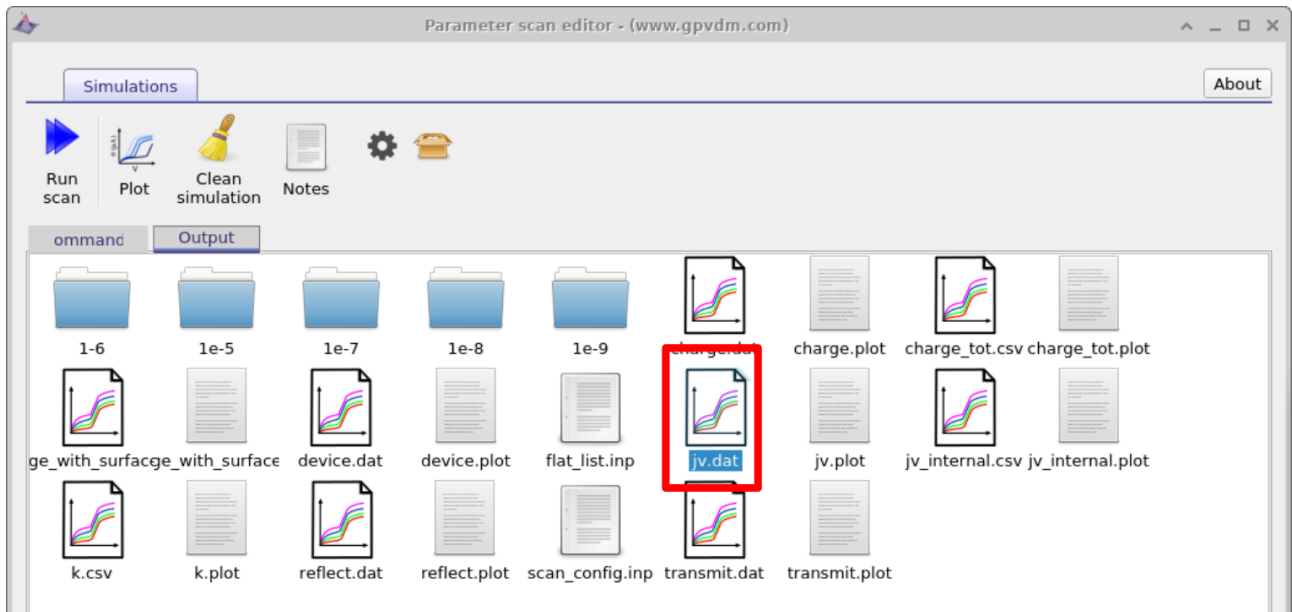


Figure 16.6: Step 7: The output tab showing the five simulation directories and the multicolored plot files.

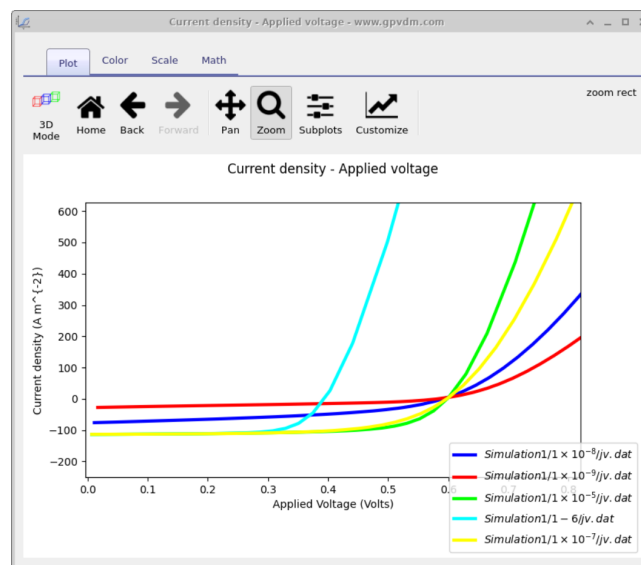


Figure 16.7: Step 8: The result of the mobility scan.

16.1.2 Duplicating parameters - changing the thickness of the active layer

Very often one wants to change a parameter, then set another parameter equal to the parameter which was changed. An example of this is one may want to change electron and hole mobilities together when simulating a device with symmetric mobilities. This can be done using the duplicate function of the scan window as seen in figure 16.8. In this example we tackle a slightly more tricky problem than changing mobilities together we are going to change the physical width of the active layer and at the same time adjust the electrical mesh to make it match. As discussed in section 7 the width of the active layer must always match the width of the electrical mesh. When you change the layer width by hand in the layer editor OghmaNano updates the width of the electrical mesh for you. But when scripting the model it won't do

this update for you. Therefore in the example below we are going to set the width of the active layer by scanning over:

epitaxy→PM6:Y6→dy of the object

Then we are going to add another line under and under parameter to scan select

mesh→mesh_y→segment0→len

and set it to

epitaxy→PM6:Y6→dy of the object

under the operation dropdown box. You will see the word duplicate appear under values.

If you now run the simulation "epitaxy→PM6:Y6→dy of the object" will be changed and "mesh→mesh_y→segment0→len" will follow it.

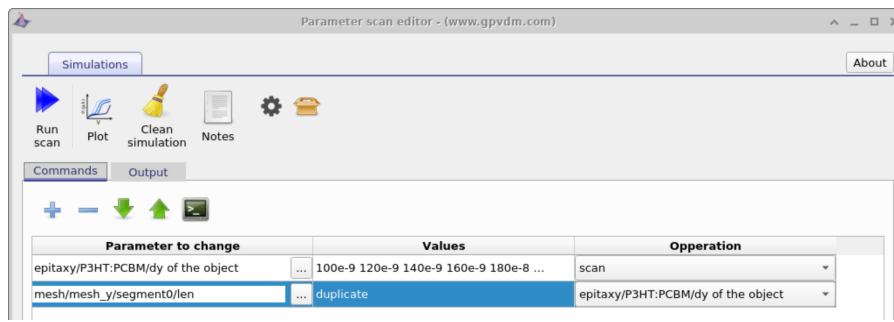


Figure 16.8: Duplicating material parameters.

Side note: Device with multiple active layers

The sum of the active layer thickness (as defined in the layer editor) MUST equal the electrical mesh thickness (more about the mesh in section 7). If for example one had three active layers TiO₂ (100 nm)/Perovskite (200 nm)/Spiro (100 nm) with a total width of 400 nm. The total mesh length must be 400 nm as well. Therefore were one want to change the thickness of the perovskite layer as in 16.8 one would have to break the electrical mesh up into three sections and make sure you were updating the mesh segment referring to the perovskite layer alone.

16.1.3 Setting constants

Often when running a parameter scan one wants to set a constant value, this can be done using the "constant" option in the Operations dropdown menu. See figure 16.9

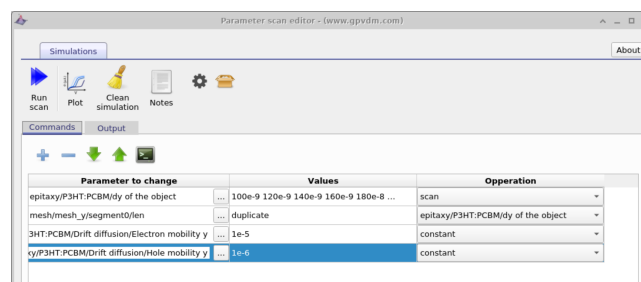


Figure 16.9: The result of the mobility scan.

16.1. THE PARAMETER SCAN WINDOW

16.1.4 The equivalent of loops

Often when scanning over a parameter range one may want to simulate so many parameters that it is not practical to type them in. In this case OghmaNano has the equivalent of a loop. So for example if one wanted to change a value from 100 to 400 in steps of 1, one could type

```
[100 400 1]
```

Listing 1: The equivalent of loops in OghmaNano, this is often quicker than typing parameters in by hand.

16.1.5 Limitations of the scan window

Although the scan window is convenient in that it provides a quick way to scan simulation parameters, it is by nature rather limited in terms of flexibility. If you want to do complex scans where multiple parameters are changed or to programmatically collect data from each simulation then you can use the or matlab interfaces to OghmaNano. These are described in the latter sections.

16.2 Multiparameter device optimizer

Related YouTube videos:



Optimizing the layer structure of a Perovskite solar cell



Optimizing an OPV device for maximum photon harvesting.



Searching for the optimum layer structure in an organic solar cell.

Very often when optimizing a device an engineer or scientists will want to know what the optimum structure of a device is. For example a perovskite solar cell is made up of multiple layers, but what is the optimum thickness of each layer? If the perovskite layer is made really thick then lots of light will be absorbed but the down side of this is that it will take longer for charge carriers to escape the device so recombination will be high. Conversely if the layer is made really thin very few carriers will have a chance to recombine as they will not spend long in the device but the downside is that not many photons will be absorbed in the first place as the layer is thin. If one then also considers that light will reflect multiple times of interfaces in the device setting up standing wave patterns, this will further complicate the optimization problem as one will need to optimize not only the thickness of the perovskite layer but also the thicknesses of all other layers at the same time. To solve this multi-parameter optimization problem one can use the *Fast optimizer* within the scan window.

16.2.1 Using the multi parameter optimizer

In the new simulation window under the sub-topic *Scripting and fitting* there are several examples of multi-parameter optimizers:

- Electrical layer optimizer: This will vary the layer thickness of two active layers of an organic solar cell simulation and plot the PEC/FF/Voc as a function of these layer thicknesses.
- Optical layer optimizer (perovskite): This will vary the thickness of two layers in a perovskite solar cell and plot the current generated by each layer within the device.
- Optical layer optimizer (OPV): This will vary the thickness of two layers in a perovskite solar cell and plot the current generated by each layer within the device.

In this text we will be using the *Optical layer optimizer (perovskite)*, if you open this simulation and navigate to the scan window, you will see a scan already set up called *optimizer*. If you open it you will get a window shown in figure 16.10. This scan window looks just like the scan windows described in the previous section, however the key difference is that the *Fast optimizer* button is depressed. When this button is depressed scan results are not written to disk, instead the key simulation parameters are tabulated and saved to disk at the end of the simulation. Notice that in this example we are varying the thickness (dy) of the Perovskite layer between 300nm and 500 nm in steps of 10 nm and the thickness (dy) TiO₂ layer from 100 nm to 300 nm also in steps of 10 nm. Try running the simulation, the using windows explorer

16.2. MULTIPARAMETER DEVICE OPTIMIZER

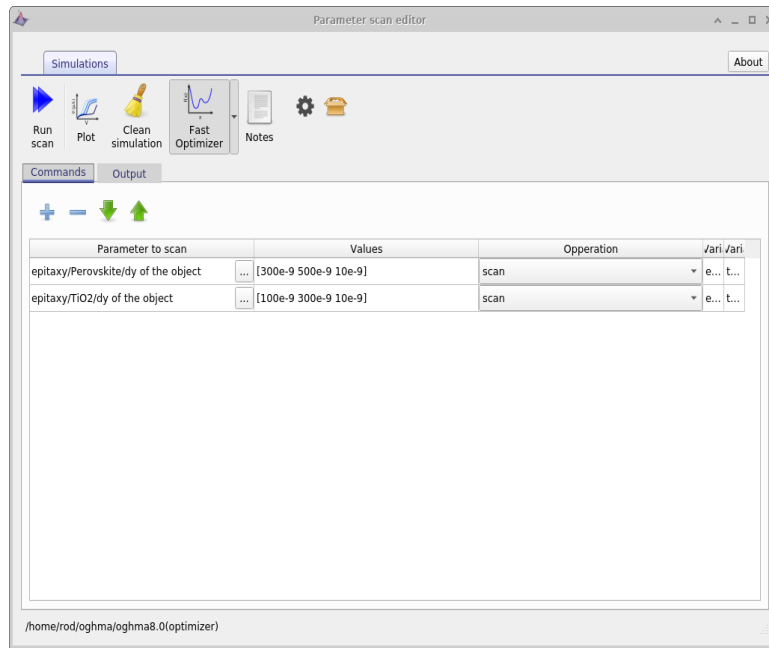


Figure 16.10: The scan window with the optimizer button depressed ready to run a device layer optimization.

navigate to your simulation directory, then open the folder called *optimize* and in there you will find a *csv* file called *optimizer_output.csv*. If you open this with Excel or LibreOffice, it will look like figure 16.11.

If you examine figure 16.11 carefully you can see the first two columns are labelled *epitaxy.layer2.dy* and *epitaxy.layer1.dy*. These are the layer thicknesses we decided to change in the scan window. For every subsequent layer in the device there are two columns, labelled *layerX/light_frac_photon_generation* and *layerX/J*. These refer to the fraction of the light absorbed within the layer and the maximum current this layer would produce if all the light absorbed within the layer were turned into current. Clearly if light is absorbed within the active layer it has a good chance of being turned into current, however if light is absorbed within the back metallic contact then there is little chance of that light being turned into electrical current. If you use the sorting tools included within Excel/LibreOffice you can figure out which device structures produce the most current.

optimizer_output.csv - LibreOffice Calc

File Edit View Insert Format Styles Sheet Data Tools Window Help

Liberation Sans 10

A1 optimizer_output.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	epitaxy.layer2.dy	epitaxy.layer1.dy	layer0/light_trac_photon_generation	layer0/J	layer1/light_trac_photon_generation	layer1/J	layer2/light_trac_photon_generation	layer2/J	layer3/light_trac_photon_generation	layer3/J	layer4/light_trac_photon_generation	layer4/J			
1	5E-08	1E-08	0.200887	69.91554	0.002569722	0.8979193	0.5976408	208.8293	0.1556977	54.40432	0.04400317	15.37571			
2	5E-08	1E-08	0.1963058	70.51342	0.002569692	0.8466357	0.6113281	219.5902	0.1475556	53.00224	0.04245346	15.24936			
3	6E-08	1E-08	0.1931103	71.32338	0.002223093	0.8210778	0.6219428	229.7085	0.1410971	52.11283	0.04162671	15.37442			
4	6E-08	1E-08	0.1931262	72.41942	0.002205584	0.8207607	0.6263728	234.8803	0.1372663	51.47277	0.04102908	15.38529			
5	7E-08	1E-08	0.1927131	73.46838	0.002267393	0.8644024	0.6342457	241.7947	0.1301837	49.63016	0.04059014	15.47425			
6	7E-08	1E-08	0.1948376	74.88562	0.002569312	0.9082677	0.6354263	244.2254	0.1269945	48.8102	0.04037848	15.51942			
7	8E-08	1E-08	0.1977635	76.48971	0.002480909	0.9595498	0.6356806	245.8645	0.1237925	47.87965	0.04028247	15.5802			
8	8E-08	1E-08	0.201609	78.36588	0.002608468	1.013918	0.6350162	246.8323	0.1204678	46.82613	0.04029652	15.94141			
9	9E-08	1E-08	0.2089747	80.94028	0.002399105	0.9232327	0.6511299	252.1962	0.09675845	37.47657	0.04073778	15.77859			
10	9E-08	1E-08	0.2153273	83.75405	0.00250137	0.9729367	0.6475977	251.8904	0.09358979	36.40283	0.04098453	15.94141			
11	1E-07	1E-08	0.2231289	87.23446	0.00258444	1.010412	0.6427212	251.2782	0.09016221	35.24981	0.0410332	16.18699			
12	1.05E-07	1E-08	0.2324536	91.49132	0.002642751	1.040159	0.6363777	250.4714	0.08649164	34.04221	0.04203504	16.54456			
13	1.1E-07	1E-08	0.2432652	96.5996	0.002672812	1.061363	0.6285168	249.5814	0.08262342	32.80942	0.04292181	17.04407			
14	1.15E-07	1E-08	0.2554177	102.5987	0.002673426	1.073888	0.619183	248.7194	0.07862275	31.58195	0.04410308	17.71575			
15	1.2E-07	1E-08	0.2686167	109.4651	0.002646067	1.07831	0.6085551	247.9948	0.07456688	30.38779	0.0456135	18.58815			
16	1.25E-07	1E-08	0.2821764	116.9508	0.002596008	1.075941	0.5971935	247.5127	0.07057213	29.24931	0.04746195	19.67107			
17	1.3E-07	1E-08	0.2917728	121.2358	0.002195487	0.9122568	0.6024717	250.3357	0.05573049	23.15683	0.04782956	19.67387			
18	1.35E-07	1E-08	0.3021246	127.1534	0.002141699	0.9039166	0.5935753	250.3033	0.05287089	22.31447	0.04980598	21.02074			
19	1.4E-07	1E-08	0.3021352	128.8591	0.002490562	1.062221	0.5803024	247.496	0.06350114	21.08291	0.05157071	21.99465			
20	1.45E-07	1E-08	0.290609	121.8266	0.002513554	1.05371	0.5927431	248.4846	0.06245583	26.18219	0.0516785	21.66421			
21	1.5E-07	1E-08	0.283698	116.5248	0.002167763	0.8903774	0.6133504	251.9248	0.05049829	20.74144	0.05028555	20.65406			
22	1.55E-07	1E-08	0.2439836	94.44813	0.002295374	0.8885588	0.6554553	253.7323	0.05191573	20.09702	0.04634987	17.94247			
23	1.6E-07	1E-08	0.1939939	70.01761	0.002476622	0.9887977	0.7096104	256.1175	0.05409127	19.523	0.03982784	14.37494			
24	1.65E-07	1E-08	0.1529745	52.7581	0.003120457	1.076188	0.7410065	255.5595	0.06939634	23.93352	0.03350213	11.55427			
25	1.7E-07	1E-08	0.1434242	48.61655	0.002714776	0.9202284	0.7667482	259.9048	0.05611271	19.02542	0.03098564	10.50321			
26	1.75E-07	1E-08	0.1234073	41.09446	0.002834039	0.9437313	0.791045	263.4169	0.0560145	18.65275	0.0266992	8.890798			
27	1.8E-07	1E-08	0.1136794	37.49762	0.00292494	0.9648033	0.8128426	268.1194	0.04712627	15.54479	0.02342679	7.727422			
28	1.85E-07	1E-08	0.1109234	36.8365	0.002972472	0.9671267	0.8079313	268.3055	0.05547889	18.42395	0.02269391	7.536408			
29	1.9E-07	1E-08	0.1058778	35.33526	0.003039401	1.014358	0.8155552	272.1802	0.05467564	18.24723	0.02085189	6.959028			
30	1.95E-07	1E-08	0.1063118	35.6137	0.003072538	1.029279	0.8156738	273.245	0.05447667	18.33975	0.02019519	6.765247			
31	1.95E-07	1E-08	0.2007803	70.86006	0.003826516	1.350467	0.595419	210.0395	0.1563432	55.17718	0.04390814	15.49621			
32	5E-08	1E-08	0.197584	71.63666	0.00360315	1.306369	0.6045617	219.1917	0.151202	54.82025	0.04304911	15.60801			
33	5E-08	1E-08	0.1954478	72.54562	0.0031666	1.175367	0.6155089	228.4625	0.1439785	53.44145	0.04189822	15.55164			
34	6E-08	1E-08	0.1942413	73.4496	0.003473036	1.313262	0.622881	235.5331	0.1379193	52.15151	0.04147826	15.6942			
35	6E-08	1E-08	0.195228	74.77217	0.003264028	1.25012	0.6283175	240.645	0.1322714	50.65985	0.04091901	15.67194			
36	7E-08	1E-08	0.19754	76.27646	0.003394734	1.310815	0.6295768	243.0996	0.1287535	49.71584	0.040735	15.72908			
37	8E-08	1E-08	0.2007746	78.02865	0.003547165	1.378563	0.6298148	244.77	0.1251895	48.6534	0.04067388	15.80741			
38	8E-08	1E-08	0.207043	80.28734	0.003722112	1.443365	0.6463053	250.625	0.101757	39.45943	0.04117264	15.96597			
39	8E-08	1E-08	0.2125226	82.83679	0.003876846	1.511112	0.6483268	250.95	0.09944968	38.37359	0.04132411	16.10726			
40	9E-08	1E-08	0.2203071	86.29795	0.003675081	1.43959	0.6407543	250.9941	0.09369027	36.70003	0.04157319	16.28491			
41	9E-08	1E-08	0.2290065	90.2991	0.003769302	1.486214	0.6351466	250.4346	0.08995487	35.46868	0.04212281	16.0878			
42	1.0E-07	1E-08	0.2382635	95.13524	0.003828249	1.522177	0.627967	249.6904	0.08601907	34.20265	0.04292224	17.06661			
43	1.1E-07	1E-08	0.2509712	100.8682	0.003848629	1.548808	0.6192255	248.8738	0.08194093	32.93299	0.04401379	17.80964			
44	1.15E-07	1E-08	0.2638922	107.4969	0.003830477	1.560351	0.6090471	248.0963	0.07779361	31.68936	0.04543661	18.50888			
45	1.2E-07	1E-08	0.2774986	114.8682	0.003778386	1.564025	0.5978342	247.4675	0.0736735	30.48641	0.04721431	19.54389			
46	1.25E-07	1E-08	0.2873531	119.1988	0.003418931	1.418229	0.6034183	250.3079	0.0581991	24.14195	0.04761058	19.74965			
47	1.3E-07	1E-08	0.2990334	126.0917	0.003345818	1.410812	0.5928352	249.9773	0.05514599	23.25309	0.04963962	20.83125			
48	1.35E-07	1E-08	0.3020138	129.0493	0.003264266	1.394808	0.577972	246.9651	0.0651256	27.82791	0.0516243	22.05886			
49	1.4E-07	1E-08	0.3049021	134.7617	0.003655027	1.546395	0.5847010	247.9648	0.06447547	27.95609	0.05293547	23.12665			

optimizer_output

Find

Sheet 1 of 1

Default

English (USA)

Average: Sum: 0

100%

Figure 16.11: The file your simulation directory/optimizer/optimizer_output.csv opened in LibreOffice (You can use Excel).

16.3 Python/MATLAB scripting of OghmaNano

Scripting offers a more powerful way to interact with gypdm. Rather than using the graphical user interface, you can use your favourite programming language to interact with OghmaNano. This gives you the option to drive simulations in a far more powerful way than can be done using the graphical interface alone. Below I give examples of using MATLAB and python to drive OghmaNano, but you can use any language you want which has a json reader/writer. Pearl and Java are two languages which spring to mind.

Before you begin scripting OghmaNano you need to tell windows where OghmaNano is installed, the default OghmaNano will be installed to C:\Program files x86 \OghmaNano, in there you will see in this directory there are two windows executables, one called *oghma.exe*, this is the graphical user interface, and a second .exe, called *oghma_core.exe*. You can run *oghma_core.exe* from the command line without *oghma.exe*. You simply need to navigate to a directory containing a *sim.oghma* folder and call *oghma_core.exe*, this can be done from the windows command line, matlab, python or any other scripting language. However, before you can do this on windows, you need to add C:\Program files x86 \OghmaNano to your windows path so that windows knows where OghmaNano is installed. An example of how to do this on a modern version of windows is given in the link [https://docs.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ee537574\(v=office.14\)](https://docs.microsoft.com/en-us/previous-versions/office/developer/sharepoint-2010/ee537574(v=office.14))

Every new version of windows seems to move the configuration options around, so you may have to find instructions for your version of windows.

16.3.1 Python scripting

Related YouTube videos:



Python scripting perovskite solar cell simulation

There are two ways to interact with .oghmafiles via python, using native python commands or by using the OghmaNanoclass structures, examples of both are given below.

The native python way

As described in section 13.1, .oghmafiles are simply json files zipped up in an archive. If you extract the sim.json file form the sim.oghmafile you can use Python's json reading/writing code to edit the .json config file directly, this is a quick and dirty approach which will work. You can then use the *os.system* call to run *oghma_core* to execute OghmaNano.

For example were one to want to change the mobility of the 1st device layer to 1.0 and then run a simulation you would use the code listed in listing 2.

If the simulation in sim.json is setup to run a JV curve, then a file called sim_data.dat will be written to the simulation directory containing paramters such as PCE, fill factor, J_{sc} and V_{oc} . This again is a raw json file, to read this file in using python and write out the value of V_{oc} to a second file use the code given in listing 3.

Using OghmaNano's built in classes for reading and writing json

OghmaNanohas a set of classes that can read in OghmaNanofiles and write them to disk. The difference between using python's native commands and the gypmd classes is that, OghmaNanowill convert the json save files to a hierarchical tree of python classes rather than leaving them as raw json. So for example using Python's native json interpreters one would write:

```

import json
import os
import sys

f=open('sim.json')          #open the sim.json file
lines=f.readlines()
f.close()
lines="".join(lines)      #convert the text to a python json object
data = json.loads(lines)

#Edit a value (use firefox as a json viewer
# to help you figure out which value to edit)
# this time we are editing the mobility of layer 1
data['epitaxy']['layer1']['shape_dos']['mue_y']=1.0

#convert the json object back to a string
jstr = json.dumps(data, sort_keys=False, indent='\t')

#write it back to disk
f=open('sim.json',"w")
f.write(jstr)
f.close()

#run the simulation using oghma_core
os.system("oghma_core.exe")

```

Listing 2: Manipulating a sim.json file with python and running a OghmaNanosimulation.

```

f=open('sim_info.dat')
lines=f.readlines()
f.close()
lines="".join(lines)
data = json.loads(lines)

f=open('out.dat',"a")
f.write(str(data["Voc"])+"\n");
f.close()

```

Listing 3: Reading in a sim_data.dat file using Python's native json reader.

```

data['epitaxy']['layer1']['shape_dos']['mue_y']=1.0

```

Listing 4: Reading in a sim_data.dat file using Python's native json reader.

```
data.epitaxy.layer[1].shape_dos.mue_y=1.0
```

Listing 5: Reading in a sim_data.dat file using Python's native json reader.

```
#!/usr/bin/env python3
import json
import os
import sys

sys.path.append('c:\Program files x86\simname\modules')
from json_root import json_root

data=json_root()
data.load("sim.json")
data.epitaxy.layer[1].shape_dos.mue_y=1.0
data.save()

os.system("coreexename.exe")
```

Listing 6: Editing sim.json files using OghmaNano's built in classes.

but using the OghmaNanointerpreter one would write

The OghmaNanoclass tree also has embedded functions for searching for objects and alike some of which are described below in listing 6.

Running OghmaNanoacross multiple cores

The scan window by default uses OghmaNano's built in job scheduler so that if you want to scan across 10 parameters and have a CPU with multiple cores, the jobs will be spread across all cores. This increases the overall speed of the simulations. You can access this API using the OghmaNano_api class, an example of how to do this is given in listing 7.


```

#!/usr/bin/env python3
import os
import sys
sys.path.append('c:\Program files x86\ \simname \ modules')

from model_api import model_api

#initialize the API
api=model_api(verbose=False)

#Use the name of the current script to determine the directory name to make
script_name=os.path.basename(__file__).split(".")[0]

#define the name of the simulation dir
scan_dir=os.path.join(os.getcwd(),script_name)

#make the simulation dir
api.mkdir(scan_dir)                                #make a new scandir

#tell the API where we are going to run the simulation
api.server.server_base_init(scan_dir)

#Loop over electron and hole mobilities.
for mue in [ 1e-5, 1e-6, 1e-7, 1e-8]:
    for muh in [ 1e-5, 1e-6, 1e-7, 1e-8 ]:
        #define the sub sim path
        sim_path=os.path.join(scan_dir,"{: .2e}".format(mue),"{: .2e}".format(muh))
        #make the directory
        api.mkdir(sim_path)

        #clone the current sim dir to the new dir
        api.clone(sim_path,os.getcwd())

        #make edit the newly generated sim.json file
        data=json_root()
        data.load(os.path.join(sim_path,"sim.json"))
        data.epitaxy.layer[1].shape_dos.mue_y=mue
        data.epitaxy.layer[1].shape_dos.muh_y=muh
        data.save()

        #Add the path to the job list
        api.add_job(path=sim_path)

#run all the jobs over multiple CPUs
api.server.simple_run()

#Generate GNU PLOT compatible files for plotting the results together.
api.build_multiplot(scan_dir,gnuplot=True)

```

Listing 7: Running jobs across multiple CPUs using python

16.3.2 MATLAB scripting

As described in section 13.1 OghmaNano simulations are stored in .json files zipped up inside a zip archive. Matlab has both a zip decompressor and a json decoder. Therefore it is straight forward to edit and read and edit .oghma files in MATLAB. You can then use MATLAB to perform quite complex parameter scans. The example script below in listing 8 demonstrates how to run multiple simulations with mobilities ranging from $1e-7$ to $1e-5$ ($m^2V^{-1}s^{-1}$). The script starts off by unzipping the sim.json file, if you already have extracted your sim.json file from the sim.oghma file you don't need these lines. The code then reads in sim.json using the MATLAB json decoder *jsondecode*. A new directory is made which corresponds to the mobility value, the sim.oghma file copied into that directory. Then *json_data.epitaxy.layer0.shape_dos.mue_y* is set to the desired value of mobility and the simulation saved using *jsonencode* and *fopen*, *fprintf*, *fclose*. The *system* call is then used to run *oghma_core.exe* to perform the simulation. Out put parameters such as J_{sc} are stored in sim_data.dat again in json format, see section 4.1.4, although this is not done in this simple script.

```

if exist("sim.oghma", 'file')==false
    sprintf("No sim.oghma file found"); %Check if we have a sim.oghma file
end

if exist("sim.json", 'file')==false
    unzip("sim.oghma") %if we don't have a sim.json file
                        %try to extract it
end

A = fileread("sim.json"); %Read the json file.
json_data=jsondecode(A); %Decode the json file

mobility=1e-7 %Start mobility
original_path=pwd %working with the current dir
base_dir="mobility_scan" %output directory name
while(mobility<1e-5)
    dir_name=sprintf("%e",mobility);
    full_path=fullfile(original_path,base_dir,dir_name) %join paths
    mkdir(full_path) %make the dir
    cd(full_path) %cd to the dir

    %Update the json mobility
    json_data.epitaxy.layer0.shape_dos.mue_y=mobility %Change mobility
                                                    %of layer0

    copyfile(fullfile(original_path,"sim.oghma"),\
        fullfile(original_path,base_dir,dir_name,"sim.oghma"))

    %now write the json file back to disk
    out=jsonencode(json_data);
    json_data
    fid = fopen("sim.json",'w');
    fprintf(fid, '%s', out);
    fclose(fid);

    %run oghma - This won't work if you have not added the oghma
    %install directory to your windows paths
    system("oghma_core.exe")

    %Multiply mobility by 10
    mobility=mobility*10;
end

%Move back to the original dir
cd(original_path)

```

Listing 8: An example of how to call OghmaNano from MATLAB

Chapter 17

Output files

In general writing to disk is slow on even the most modern of computers with an SSD. The seek speed of mechanical disks has increased little of their history. Thus often writing the output data to the hard disk is the most time consuming part of any simulation. By default OghmaNanowrites all output files to disk this is so the new user can get a feel for what output OghmaNanocan provide. However to speed up simulations you should limit how much data is written to disk. The simulation editor windows (steady state,time domain etc..) offer options to decide how much data you want to dump to disk. This is shown in figure 17.1

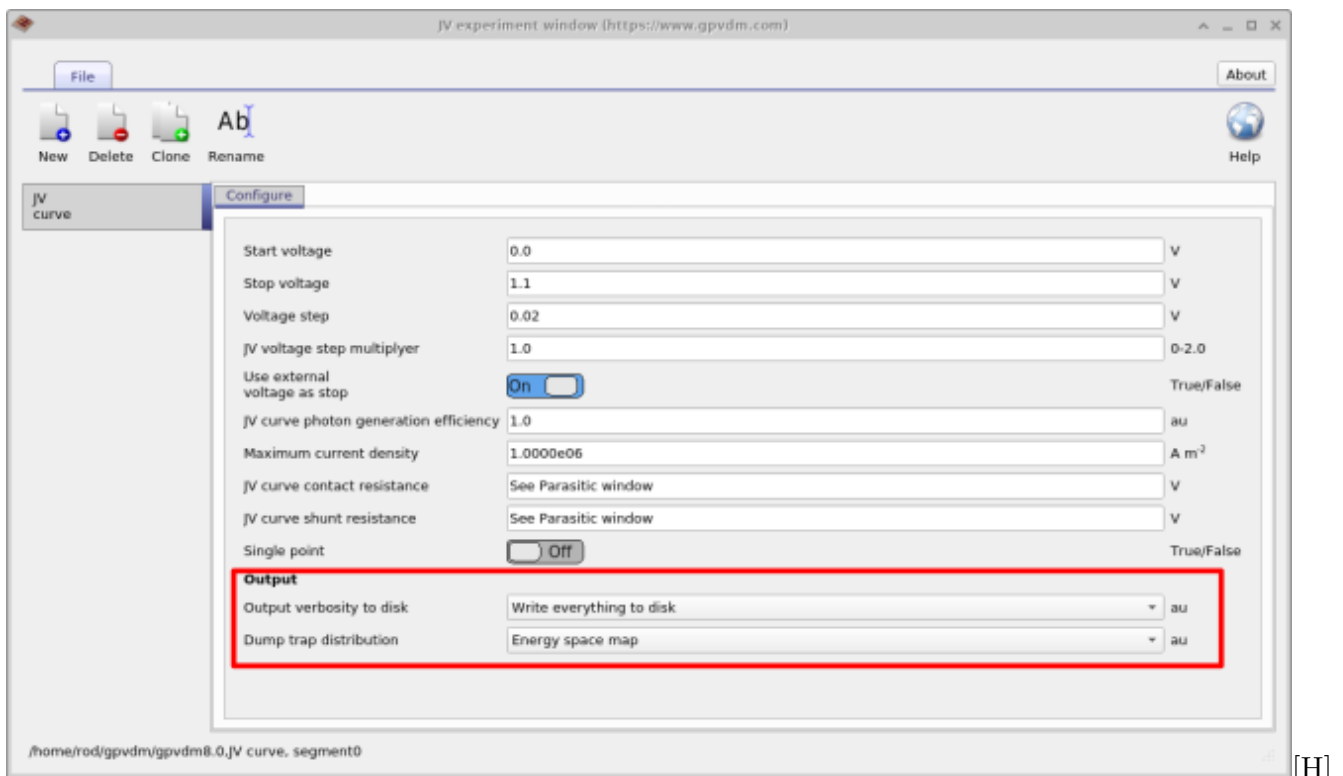


Figure 17.1: Selecting which output files are written to disk.

The option "Output verbosity to disk" can be toggled between "None" and "write everything to disk". When "None" is selected nothing is outputted to disk at all - even simulation results are not written. When "write everything to disk" is selected the simulation dumps everything to disk, so JV curves and all internal variables of the solver are written to disk so that the user can examine how carrier densities, fermi-levels, potentials etc.. change during the course of the simulation (see section 17.1). The second option below "Output verbosity to disk" called "dump trap distribution" will write out the distribution of traps in energy and position space.

See section 17.2.

17.1 Snapshots directory - dir

The snapshots directory (see figure 17.2) allows the user to plot all internal solver parameters. For example figure 17.3 where the snapshots tool is being used to plot the conduction band, valance band and quasi Fermi-levels as a function of voltage. The slider can be used to view different voltages.

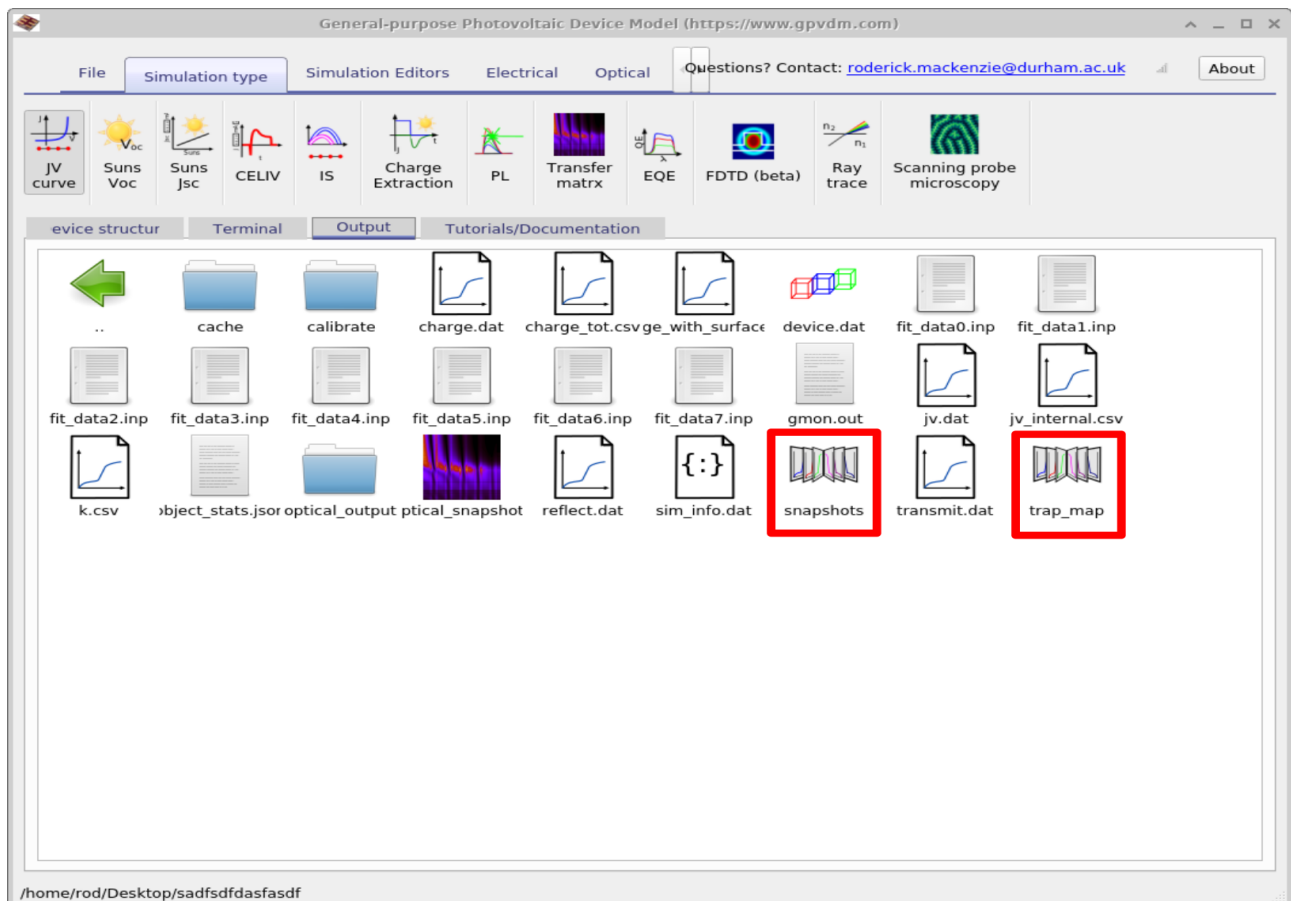


Figure 17.2: The file viewer showing the snapshots and trap map directory

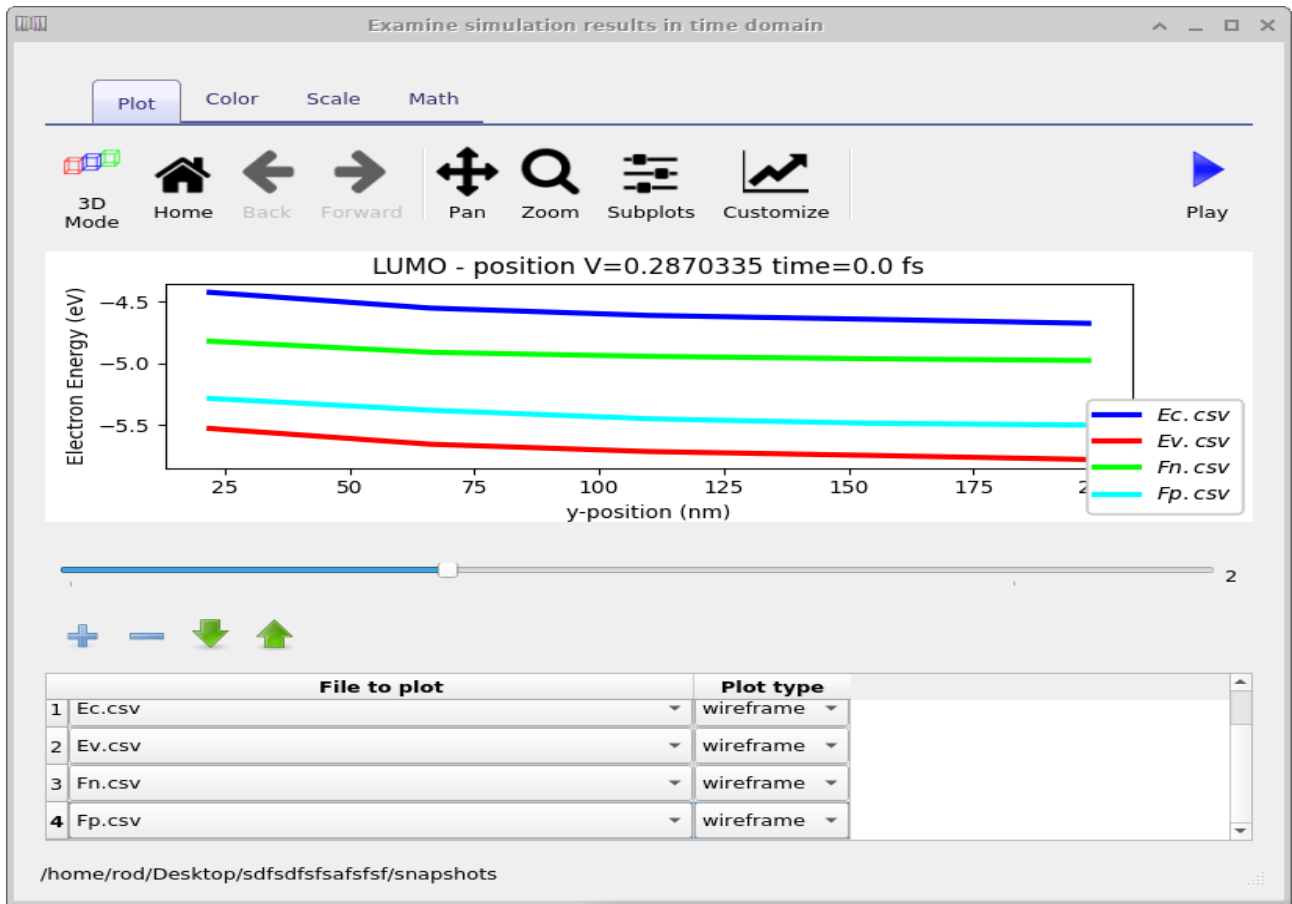


Figure 17.3: Using the snapshots tool to view the conduction band, valance band and quasi Fermi-levels

17.2 Trap_map directory - dir

The trap map directory contains the distribution and density of carriers in the traps as a function of position and energetic depth. An example is given in figure 17.4

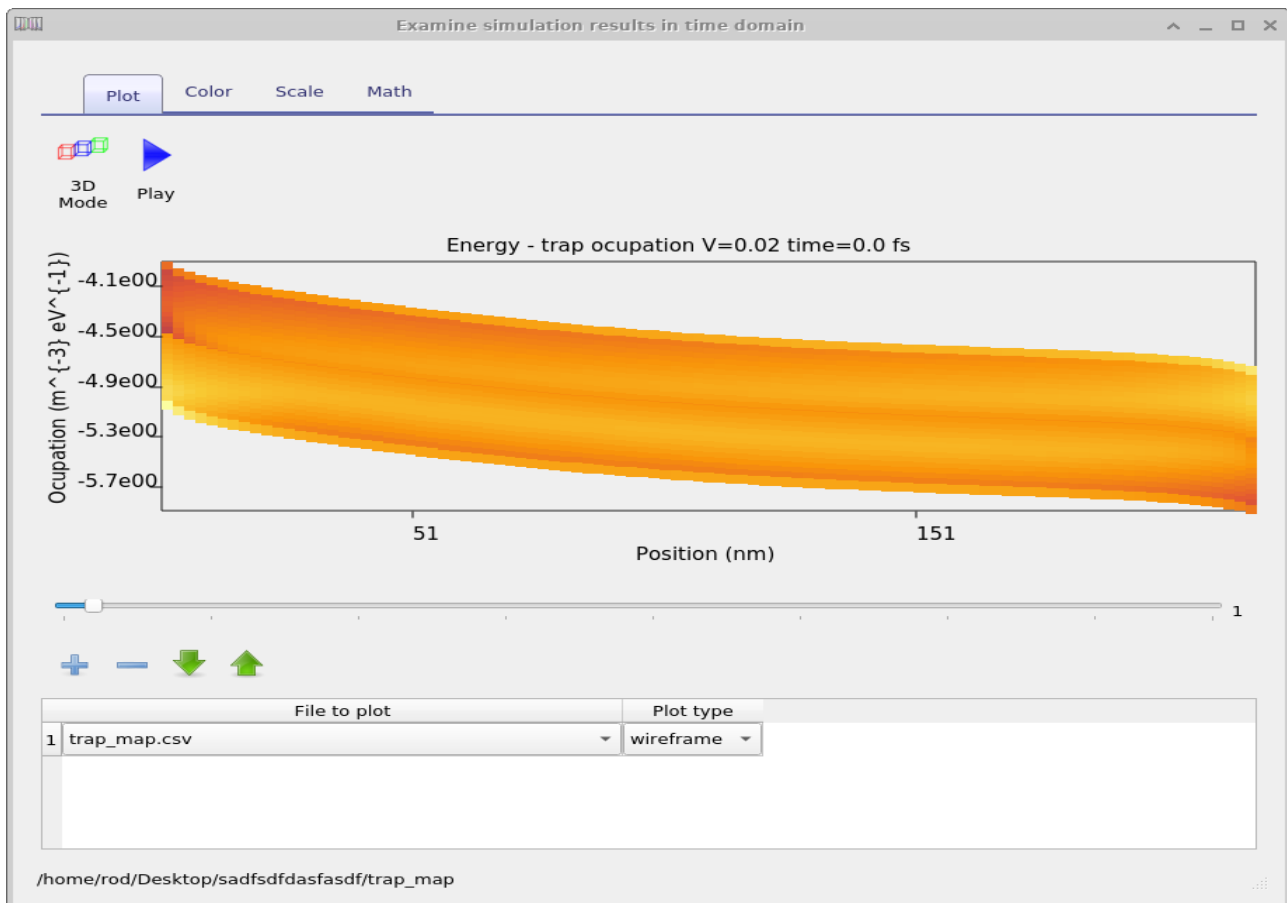


Figure 17.4: Plotting the position and energy dependence of carriers using the trap map tool

17.3 Optical snapshots - dir

Contains results of the optical simulations.

17.4 Cache - dir

Getting a computer to do math is on the whole a slow thing to do. It's much faster to precalculate results then store the answers in a look up table. This can speed up calculations significantly. The cahce dir stores the results of such precalculations, you can delete if you want it OghmaNanowill just remake it when it runs.

17.5 Equilibrium directory

Before the solver starts any simulation it solves the device equations in the dark with 0V applied bias. The result of this calculation are placed in this directory. The practical reason for doing this is that Newton's method only works if you give it a reasonable starting guess for any given problem. Thus to start the solver, we guess the carrier densities at 0V in the dark, we then use Newton's method to calculate the exact carrier density profiles at 0V in the dark (results are stored in the equilibrium directory), then from this point we can work our way to other solutions say at +1V in the light.[14]

17.5.1 Optical simulation

JSON token	Meaning	Units	Ref
J_{photo}	Photo current density Am^{-2}		
I_{photo}	Photo current A		

17.6 File formats

Almost all input and output files associated with OghmaNano are human readable, meaning that they are straight up text files. All output files can be directly plotted in gnuplot/excel as can the input files. Output files are currently called .dat, but they are simply text files. All configuration files are in json format so can be edit directly or by using the python json library.

17.6.1 .dat files

This type of file is a straight text file which can be imported into excel or any other plotting program. It contains two columns of data x and y. There is also a preamble in the file containing information such as units etc.. OghmaNano is moving from .dat files to .csv files.

17.6.2 .csv files

This is a straight csv file as you would expect which can be imported into any text editor. The first line of the file is a json string containing information such as units etc. You can ignore this. The second line of the file describes the x/y data in a human readable form then the rest of the file contains the data.

17.6.3 Binary .csv files - files which are not human readable

In some cases it is not practical to dump text files. Examples are when dealing with 3D structures. In this case OghmaNano will dump the same json header as used in the csv file but then dump a series of C floats representing the data.

Chapter 18

Troubleshooting

18.1 Windows gives warm me the software is unsigned

18.2 Why don't I get a 3D view of the device

If your simulation window looks like figure 18.2 and not like figure 18.1. It means either you do not have any 3D acceleration hardware on your computer, or you do not have the drivers for it installed. If you have an ATI/Nvidia/Intel graphics card check that the drivers are installed. Currently, not having working 3D hardware will not affect your ability to perform simulations. This is not a OghmaNanobug it's a driver/hardware issue on your computer.

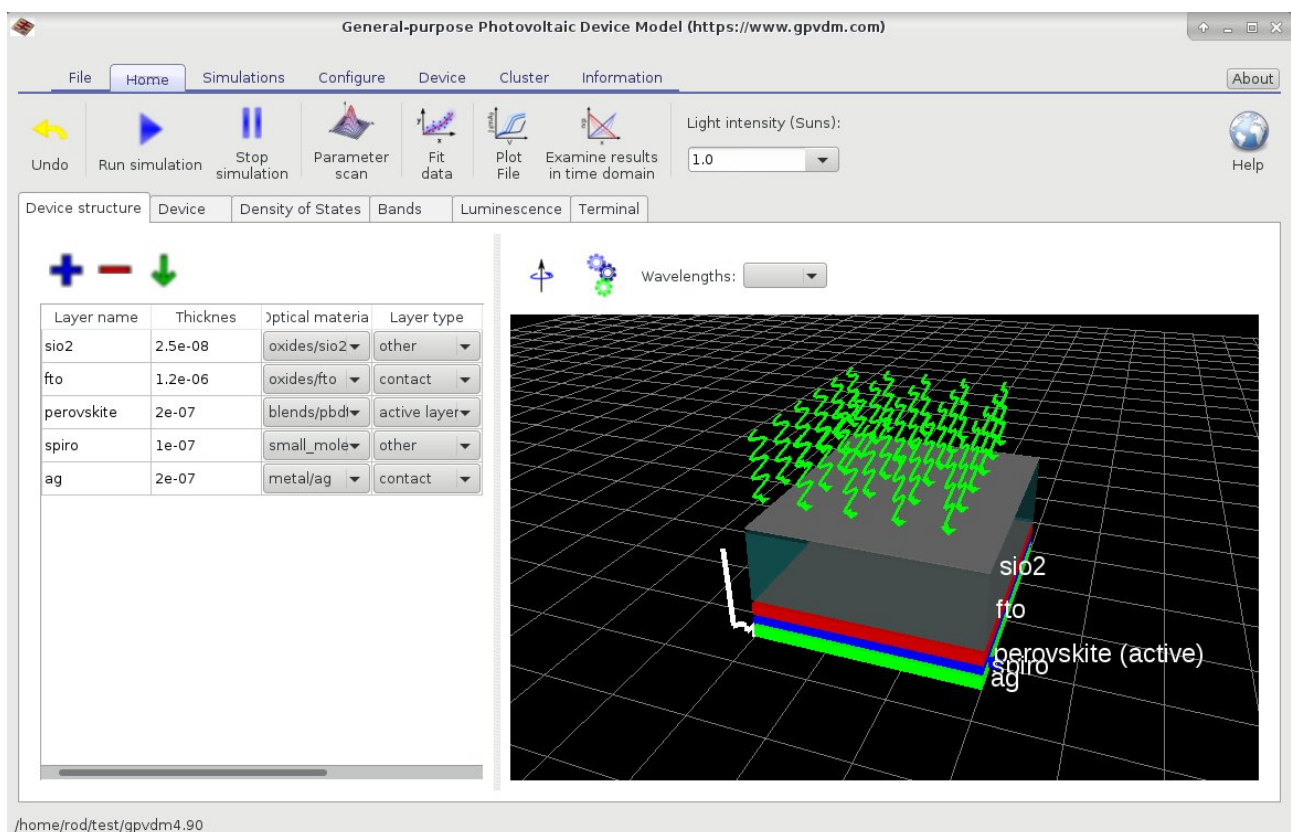


Figure 18.1: OghmaNanowith working 3D acceleration hardware.

18.2. WHY DON'T I GET A 3D VIEW OF THE DEVICE

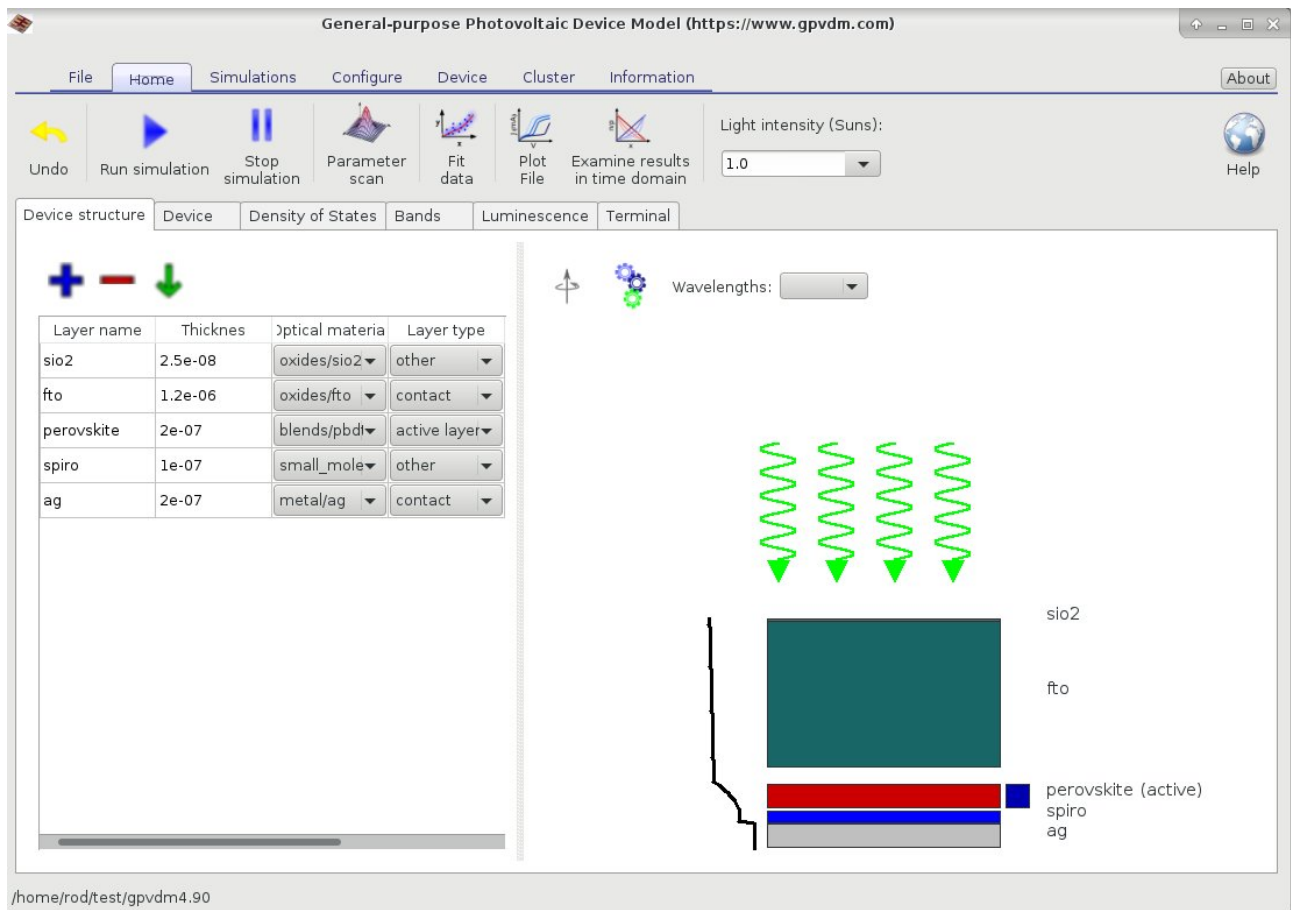


Figure 18.2: OghmaNanowith no 3D acceleration hardware.

Chapter 19

FAQ

19.1 Section

19.1.1 Should I trust the results of OghmaNano?

Yes! The model it's self has been verified against experiment [there are over 20 publications doing this, in steady state, time domain (us-fs time scales), and fx-domain]. The basic drift-diffusion solver was cross checked and compared against other drift diffusion models, and the accuracy compared down to 6-9 dp. While the optical model has been compared to analytical solutions of Maxwell's equations. The SRH model has also been compared against analytical models. If the answers you are getting out of OghmaNano are odd, then I would suggest to take a look at the input parameters. If your efficiencieis are high, try increasing the number of trap states, the recombination cross sections or reducing the e/h mobilites. Finally, I would also recommend always running the latest version, and keeping an eye on the twitter stream for bug announcements.

19.1.2 Can I use the model to simulate my exotic* material system/contacts?

The short answer is yes. The model is an effective medium model, meaning that it does not simulate the details of the medium, rather it approximates the medium with a set of electrical parameters. For example, when simulating an organic solar cell, it does not simulate every detail of the BHJ, rather it just assumes an effective mobility, density of states, recombination cross sections, trapping cross sections and so on... So if you can find electrical parameters to aproximate your material system (or guess them), there is nothing stopping you using OghmaNano to simulate any exotic device/material. The same goes for the contacts, the model simulates the contacts simply as a charge density. So if you have fancy graphene contacts which inject lots of charge, use a high majority carrier density on the contacts. Where as if you have some dirty old ITO contacts may be drop the majority carrier density a bit.

19.2 Excited states

This is a new feature and is not yet finished. I am writing the equations as I implement them in the solver.

$$\frac{dN_s}{dt} = \frac{1}{4}R_{free} + \frac{1}{4}\kappa_{TT}N_T^2 - (\kappa_s + \kappa_{ISC})N_s - \left(\frac{7}{4}\kappa_{SS}N_s - \kappa_{ST}N_T\right)N_s \quad (19.1)$$

$$\frac{dN_t}{dt} = \frac{3}{4}R_{free} + \kappa_{ISC}N_s + \frac{3}{4}\kappa_{SS}N_s^2 \quad (19.2)$$

$$\frac{dN_{SD}}{dt} = \frac{\kappa_{FRET}}{N_{DOP}}(N_{DOP} - N_{SD} - N_{TD})N_s + \frac{1}{4}\kappa_{TTD}N_{TD}^2 - \left(\frac{7}{4}\kappa_{SSD}N_{SD} + \kappa_{STD}N_{TD}\right)N_{SD} \quad (19.3)$$

Chapter 20

Legal

20.1 License

OghmaNanocomprises of three independent components, OghmaNanogui, OghmaNanocore and OghmaNanodata. In general everything is under the MIT license except the Python GUI which I have released under GPL v2.0. Details can be found [here](#).

20.2 Copyright of the manual

This manual is released under CC-BY license.

20.3 Data privacy statement

In some versions of OghmaNano it will ask you to register before using it. In these versions it asks for your name, title, company that you work for and what you intend on using OghmaNano for. This data is then transmitted to the OghmaNano server where it is securely stored. The reason I ask for this information is to be able to generate accurate usage information. Having accurate information helps when requesting grants from funding bodies. It's much easier to ask a funding body for money if you can prove you actually have users and your software is a benefit to society. Periodically OghmaNano will also contact the OghmaNano server to see if there are any software updates. By using OghmaNano you agree for the above to happen.

Bibliography

- [1] Z. Liu, Z. Deng, S. J. Davis, C. Giron, and P. Ciais. *Nature Reviews Earth & Environment*, Mar 2022.
- [2] S. Manabe and R. T. Wetherald. *Journal of Atmospheric Sciences*, 1967, 24 3 241 – 259.
- [3] R. C. MacKenzie, C. G. Shuttle, M. L. Chabiny, and J. Nelson. *Advanced Energy Materials*, 2012, 2 6 662–669.
- [4] L. Zhu, M. Zhang, J. Xu, C. Li, J. Yan, G. Zhou, W. Zhong, T. Hao, J. Song, X. Xue, et al. *Nature Materials*, 2022, 21 6 656–663.
- [5] C. Wopke, C. Gohler, M. Saladina, X. Du, L. Nian, C. Greve, C. Zhu, K. M. Yallum, Y. J. Hofstetter, D. Becker-Koch, et al. *NATURE COMMUNICATIONS*, 2022, 13 1 .
- [6] J. Piprek. *Semiconductor optoelectronic devices: introduction to physics and simulation*. Elsevier, 2013.
- [7] B. C. O’Regan, J. R. Durrant, P. M. Sommeling, and N. J. Bakker. *The Journal of Physical Chemistry C*, 2007, 111 37 14001–14010.
- [8] P. Kaienburg. 2019.
- [9] P. Kaienburg, U. Rau, and T. Kirchartz. *Phys. Rev. Applied*, Aug 2016, 6 024001.
- [10] R. C. MacKenzie, C. G. Shuttle, G. F. Dibb, N. Treat, E. von Hauff, M. J. Robb, C. J. Hawker, M. L. Chabiny, and J. Nelson. *The Journal of Physical Chemistry C*, 2013, 117 24 12407–12414.
- [11] P. Calado, A. M. Telford, D. Bryant, X. Li, J. Nelson, B. C. O’Regan, and P. R. Barnes. *Nature communications*, 2016, 7 1 1–10.
- [12] E. M. Azoff. *Solid State Electronics*, September 1987, 30 9 913–917.
- [13] S. Solak, S. Shishegaran, A. C. Hübler, and R. C. MacKenzie. *Solar RRL*, 2021, 5 12 2100787.
- [14] T. Zhan, X. Shi, Y. Dai, X. Liu, and J. Zi. *Journal of Physics: Condensed Matter*, 2013, 25 21 215301.

BIBLIOGRAPHY

Use the power of device simulation to understand your experimental data from thin film devices such as Organic Solar cells, sensors, OFET, OLEDs, Perovskite solar cells, and many more. Unlike many other models OghmaNano is purpose built from the ground up for simulating thin film devices made from disordered materials. Downloaded more than 25,000 times with over 200 papers published using the model, OghmaNano has become one of the key device modelling tools used by the scientific community developing the next generation of opto-electronic devices.

This book written by the author of OghmaNano explains will take you from a standing start to being able to confidently simulate your own devices. Learn how to simulate, Organic solar cells, Organic Field Effect Transistors, Perovskite solar cells, Organic LEDs, Large area printed devices and many more..