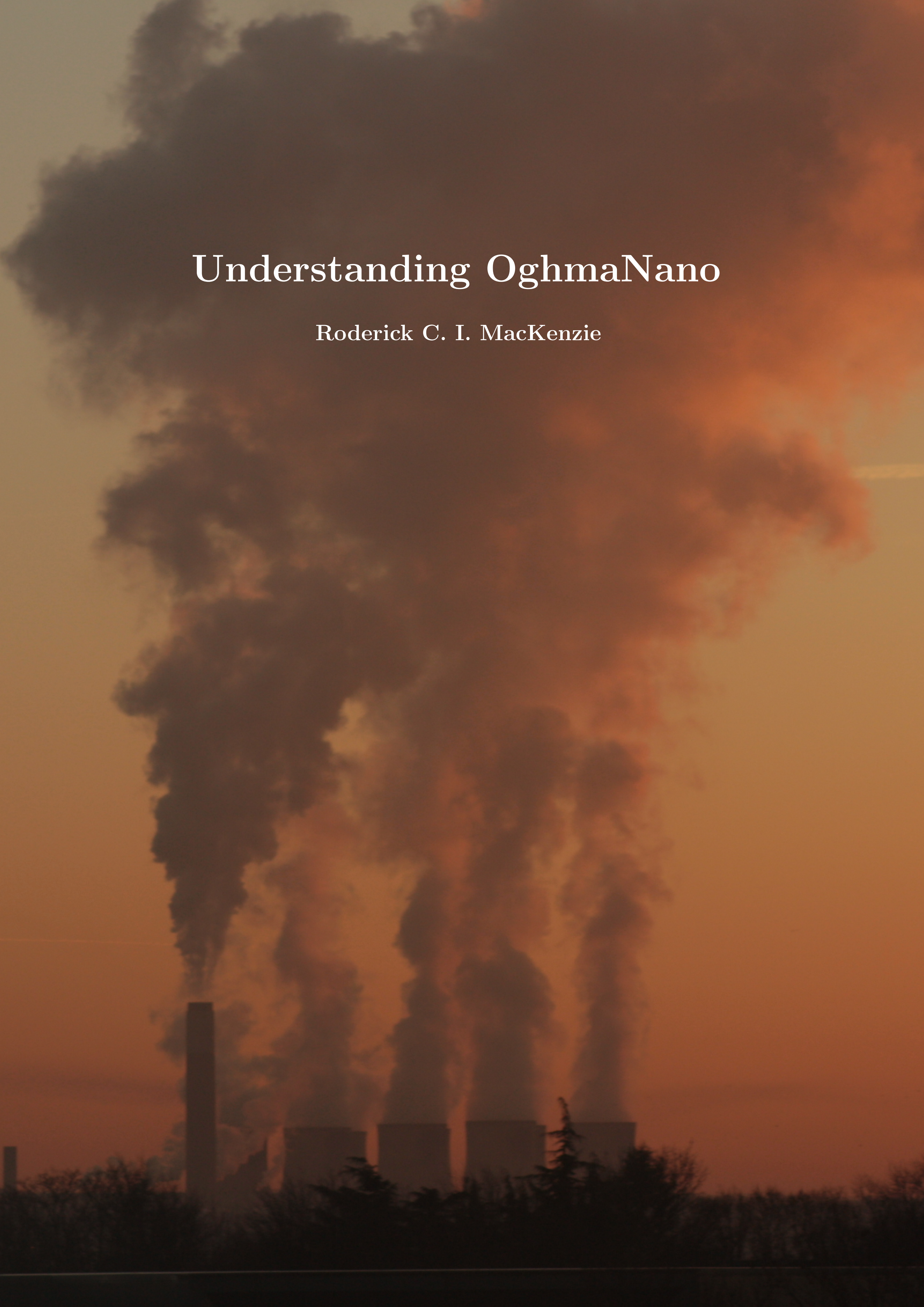


Understanding OghmaNano

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Please do not cite this manual.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 form 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.

Contents

Chapter 1

Introduction

1.1 What is OghmaNano?

OghmaNano officially stands for ***Org****anic and* ***h****ybrid* ***M****aterial* ***N****ano Simulation tool.* Oghma is also the name of the Gaelic God who's appearance is described as "sun-faced" or "shining/radiant", he is creditied 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.

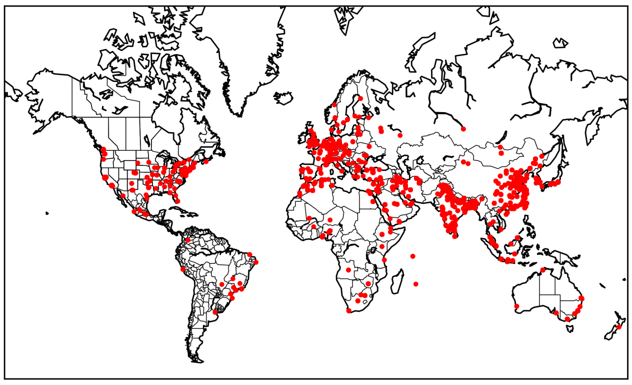


Figure 1.1: A map of locations where OghmaNano has been downloaded.

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 knolage. 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.

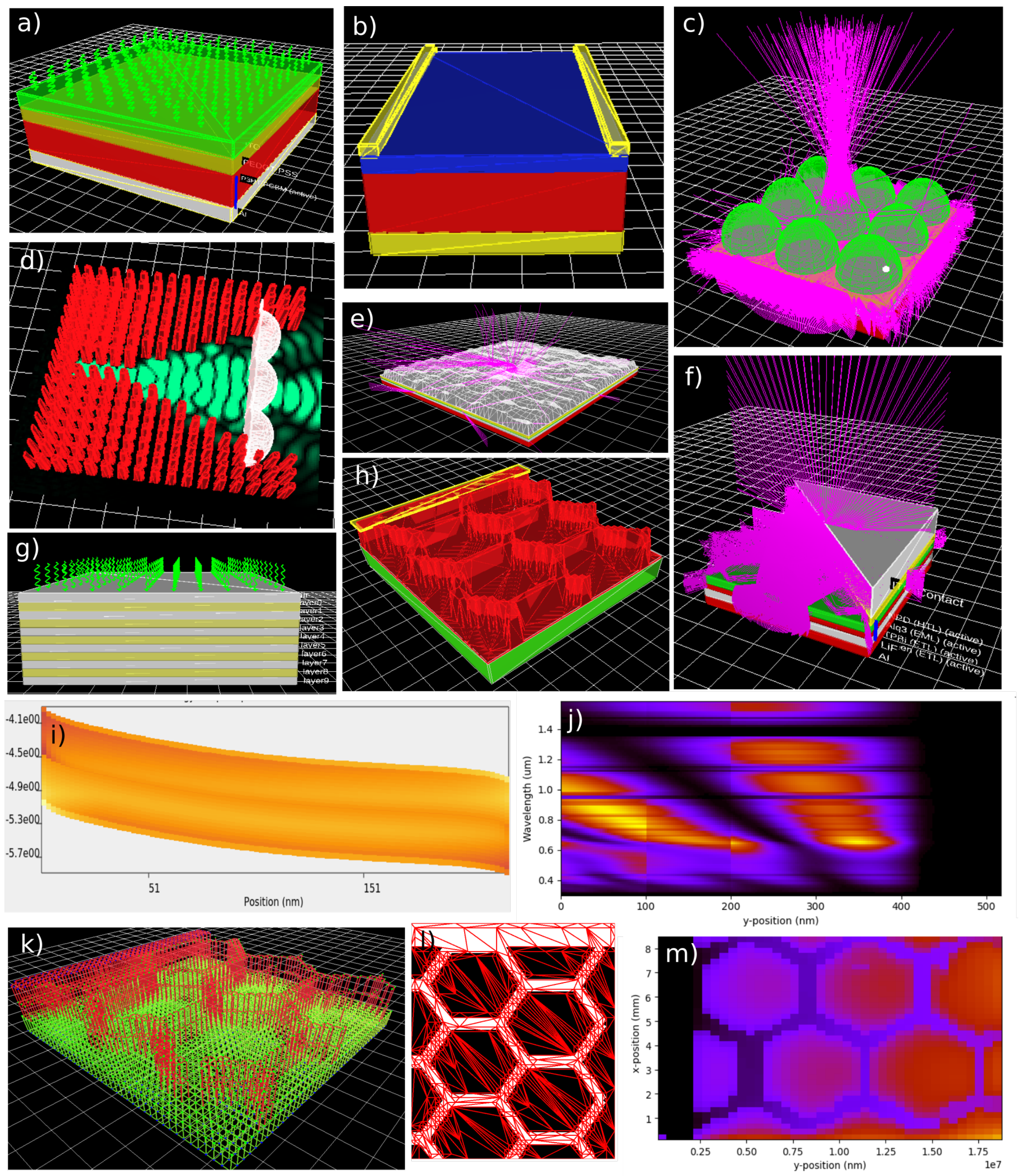


Figure 1.2: a); Organic/Perovskite solar cell simulation; b) OFET transistor simulation; c) Microlens simulation; d) Photonic waveguide simulation; e) light escaping structured films; f) OLED simulation; g) Optical filter simulation; h) large area device modelling (hexogonal contacts); i) Mapping carrier 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:

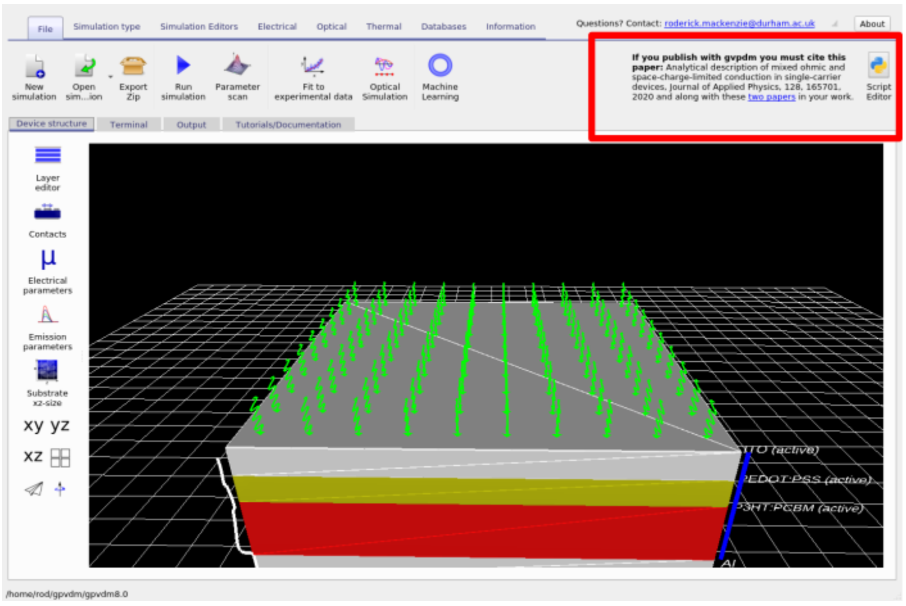
1. If you use OghmaNano to generate results, then clearly say so in your work. This can be as simple as one sentences saying: "we used OghmaNano to perform the simulations"

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:


Questions? Contact: roderick.mackenzie@durham.ac.uk

If you publish with gvpdm you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

/home/rod/gpvdm/gpvdm8.0

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).

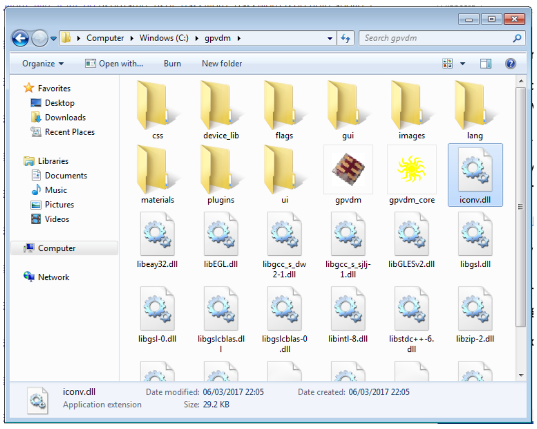


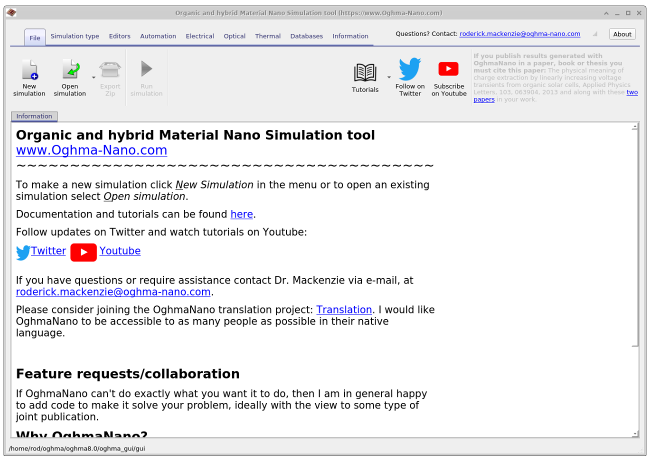
Figure 2.1: Running and installing OghmaNano. Double click on the OghmaNano icon to run the model.

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.



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: The physical meaning of charge extraction by linearly increasing voltage transients from organic solar cells, Applied Physics Letters, 103, 063904, 2013 and along with these two papers in your work.

Organic and hybrid Material Nano Simulation tool
www.Oghma-Nano.com
~~~~~~

To make a new simulation click *New Simulation* in the menu or to open an existing simulation select *Open simulation*.

Documentation and tutorials can be found here.

Follow updates on Twitter and watch tutorials on Youtube:

Twitter Youtube

If you have questions or require assistance contact Dr. Mackenzie via e-mail, at roderick.mackenzie@oghma-nano.com.

Please consider joining the OghmaNano translation project: Translation. I would like OghmaNano to be accessible to as many people as possible in their native language.

Feature requests/collaboration

If OghmaNano can't do exactly what you want it to do, then I am in general happy to add code to make it solve your problem, ideally with the view to some type of joint publication.

Why OghmaNano?

/home/rod/oghma/oghma8.0/oghma_gui/gui

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.
~~~~~~

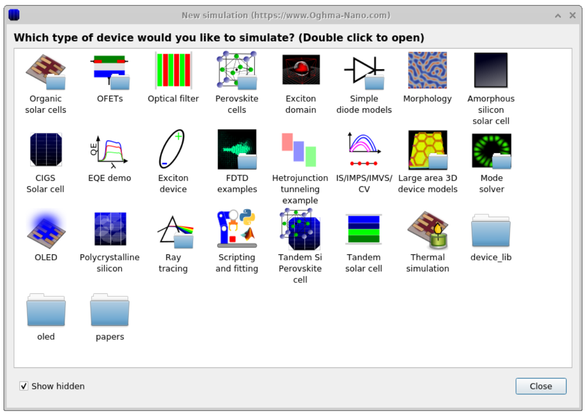


Figure 3.2: New simulation window, from here you can select different example simulations. It is often easier to start from a base simulation rather than build your own from scratch.

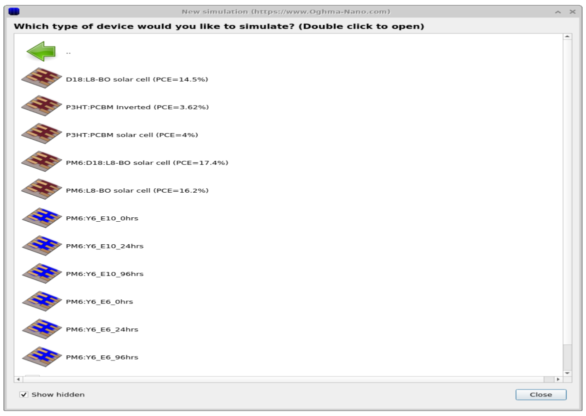
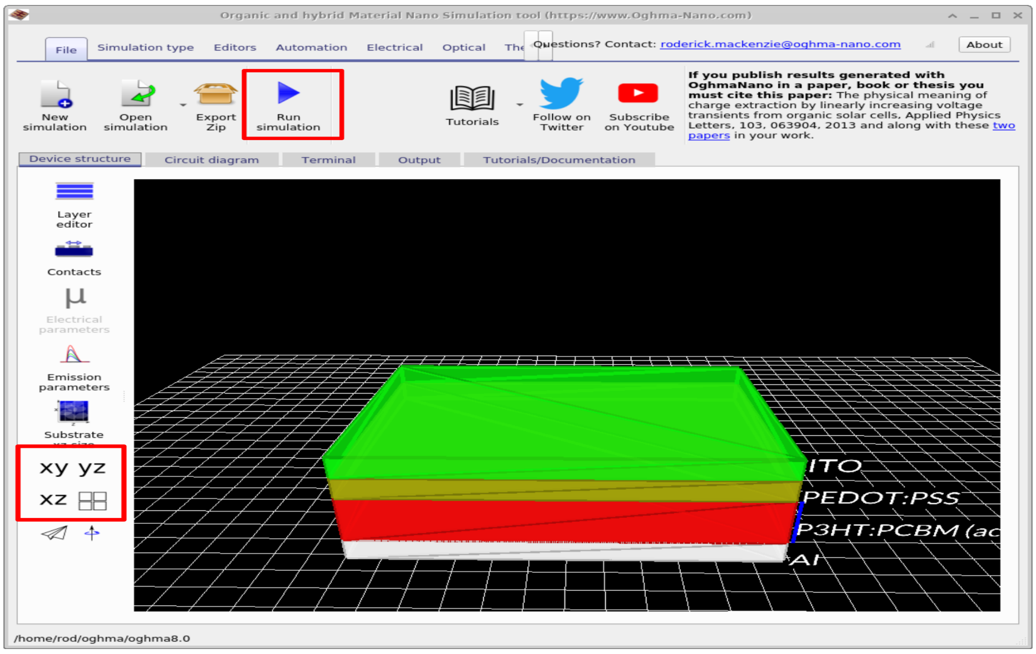

New simulation (https://www.Oghma-Nano.com)
Which type of device would you like to simulate? (Double click to open)
..
D18:L8-BO solar cell (PCE=14.5%)
P3HT:PCBM Inverted (PCE=3.62%)
P3HT:PCBM solar cell (PCE=4%)
PM6:D18:L8-BO solar cell (PCE=17.4%)
PM6:L8-BO solar cell (PCE=16.2%)
PM6:Y6_E10_0hrs
PM6:Y6_E10_24hrs
PM6:Y6_E10_96hrs
PM6:Y6_E6_0hrs
PM6:Y6_E6_24hrs
PM6:Y6_E6_96hrs
Show hidden
Close

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.



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: The physical meaning of charge extraction by linearly increasing voltage transients from organic solar cells, Applied Physics Letters, 103, 063904, 2013 and along with these two papers in your work.

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

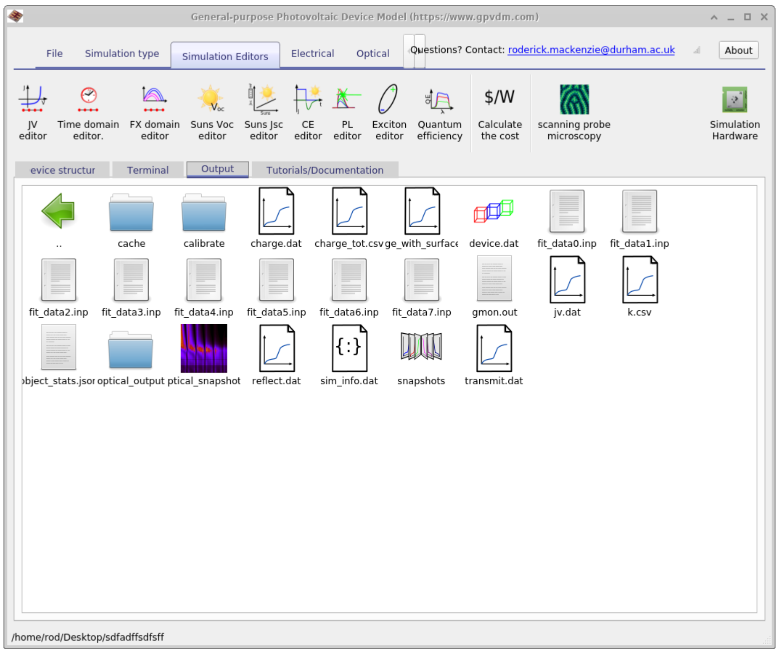


Figure 3.5: The *Output* tab this is just like windows file explorer, you can explore the simulation directory tree.

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

File name	Description
jv.dat	Current v.s. voltage curve
charge.csv	Voltage v.s. charge density curve
device.dat	The 3D device model
$fit\_data * .inp$	Experimental data for this device.
k.csv	Voltage v.s. Recombination constant k
$reflect.csv$	Optical reflection from device
transmit.csv	Optical transition through device
snapshots	Electrical snapshots see 19.1
optical_snapshots	Optical snapshots see 19.3
$sim\_info.dat$	Calculated V_{oc}, J_{sc} etc.. see 4.1.4
cache	Cache see 19.4

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.

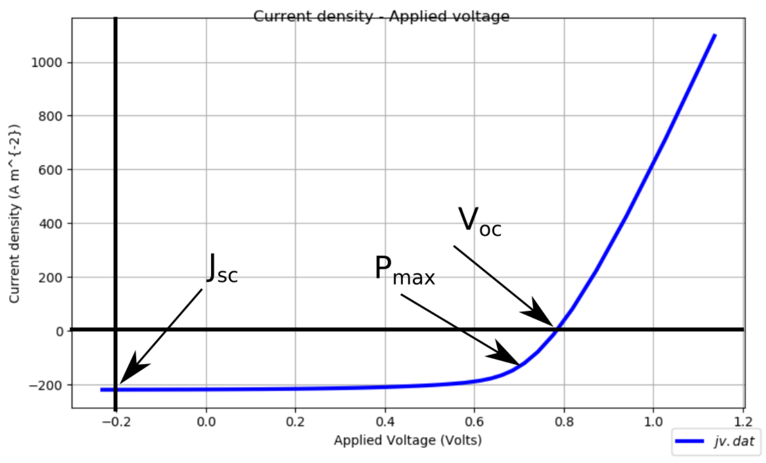
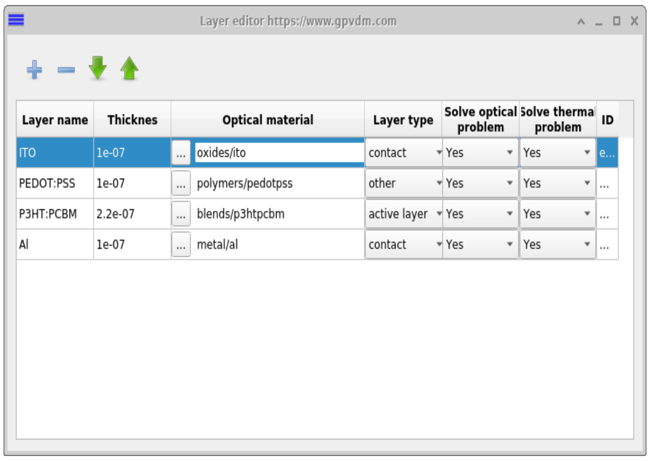


Figure 3.6: The output tab

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 iluminated), 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


Layer editor https://www.gpvdm.com

Layer name	Thicknes	Optical material	Layer type	Solve optical problem	Solve therma 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 15.1 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 21.2. 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 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

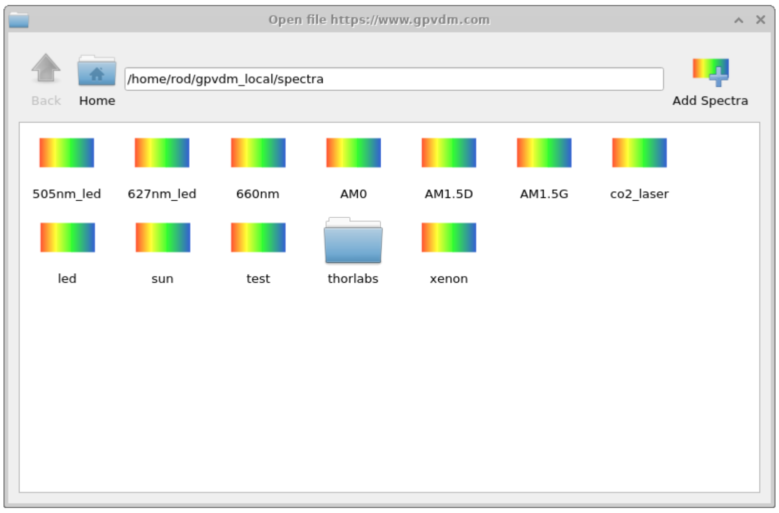
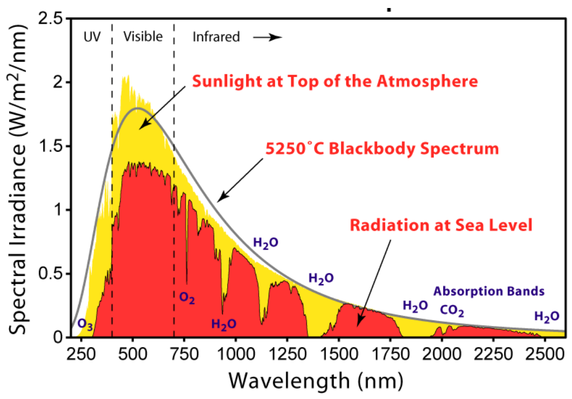


Figure 3.8: The optical database viewer

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.



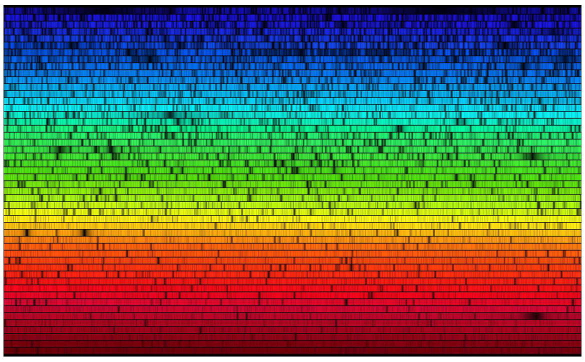


Figure 3.9: a: A plot of the entire solar spectrum. b: The image below shows the solar spectrum at 392 nm (blue) to 692 nm (red) as observed with the Fourier Transform Spectrograph at Kitt Peak National Observatory in 1981. R. Kurucz

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.

3.1.5 Light inside solar cells

As you will have seen from when you fist 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, the 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.

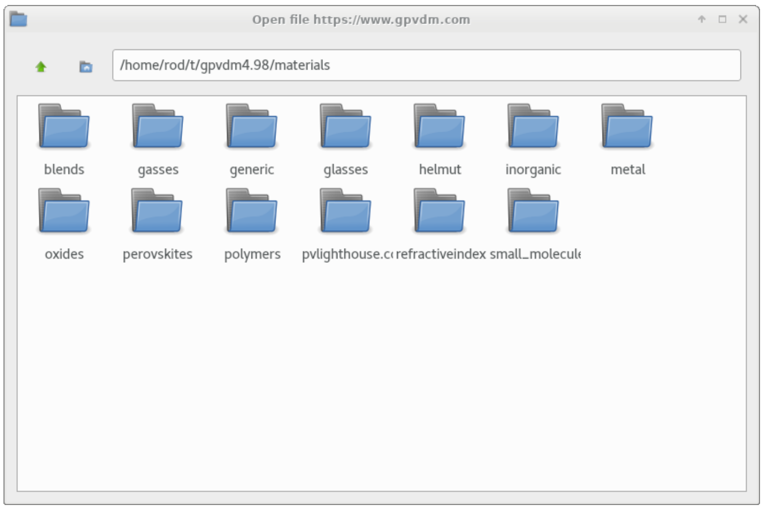


Figure 3.10: The materials database

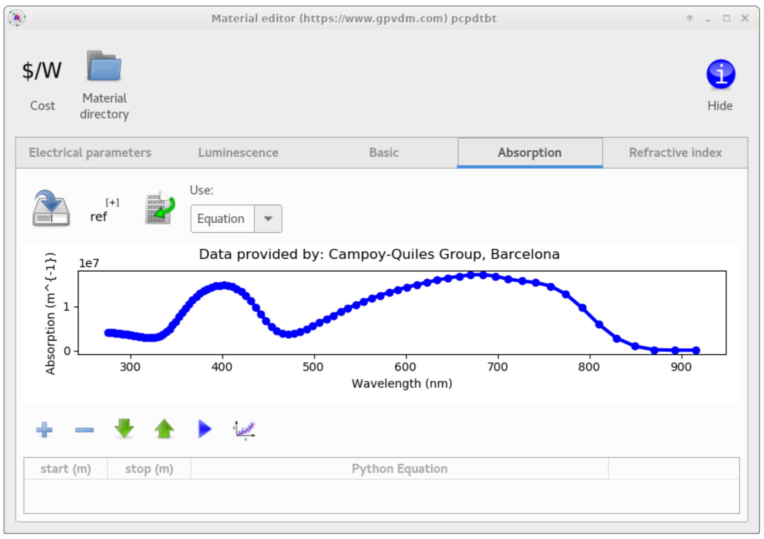


Figure 3.11: Optical absorption of the light.

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

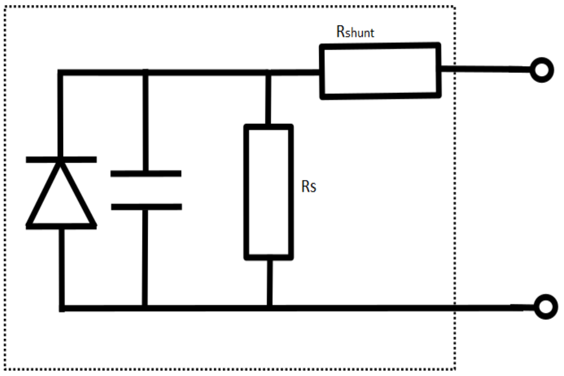
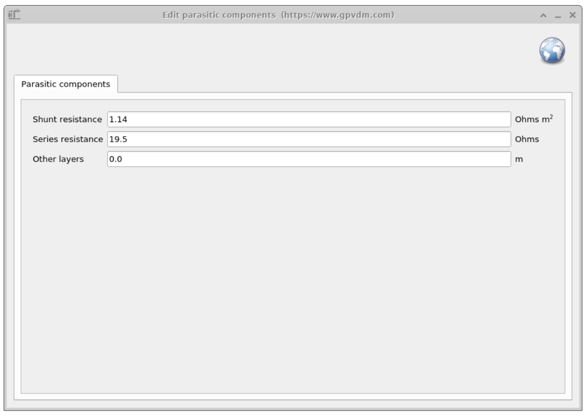


Figure 3.12: Circuit model of a solar cell.



Shunt resistance 1.14 Ohms m^2
Series resistance 19.5 Ohms
Other layers 0.0 m

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:

$$C = \frac{\epsilon_r \epsilon_0 A}{d + \Delta} \tag{3.1}$$

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 $1x10^6$ $Ohm\ m^2$ (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.

3.1.7 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.

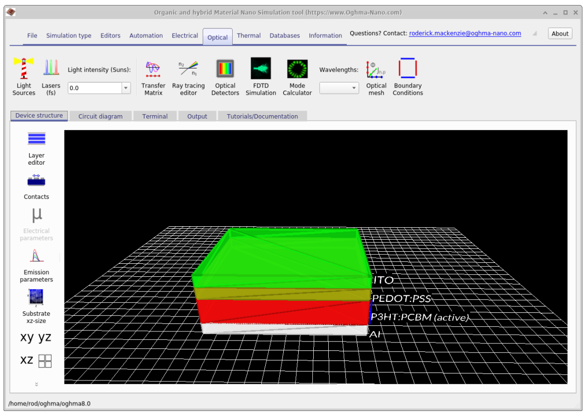


Figure 3.14: Running OghmaNano in the dark, the Light intensity drop-down menu has been set to 0 Suns and the photons have disappeared from the image.

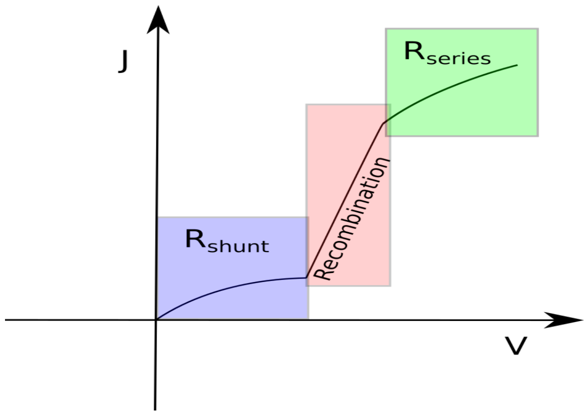


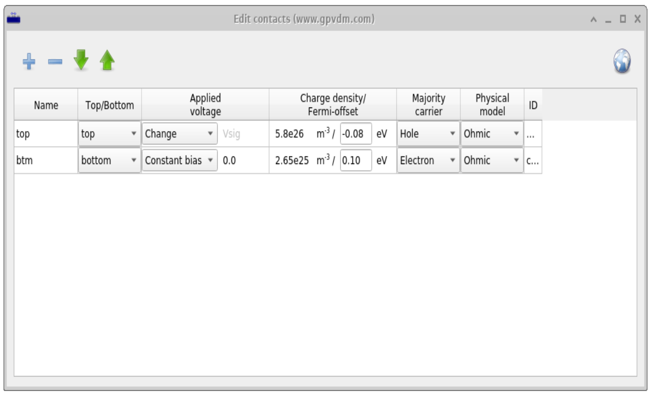
Figure 3.15: A sketch of a typical dark JV curve.

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:


Edit contacts (www.gpvdm.com)

Name	Top/Bottom	Applied voltage		Charge density/ Fermi-offset		Majority carrier	Physical model	ID
top	top	Change	Vsig	5.8e26 m^{-3} /	-0.08 eV	Hole	Ohmic	...
btm	bottom	Constant bias	0.0	2.65e25 m^{-3} /	0.10 eV	Electron	Ohmic	c...

Figure 3.16: The contact editor

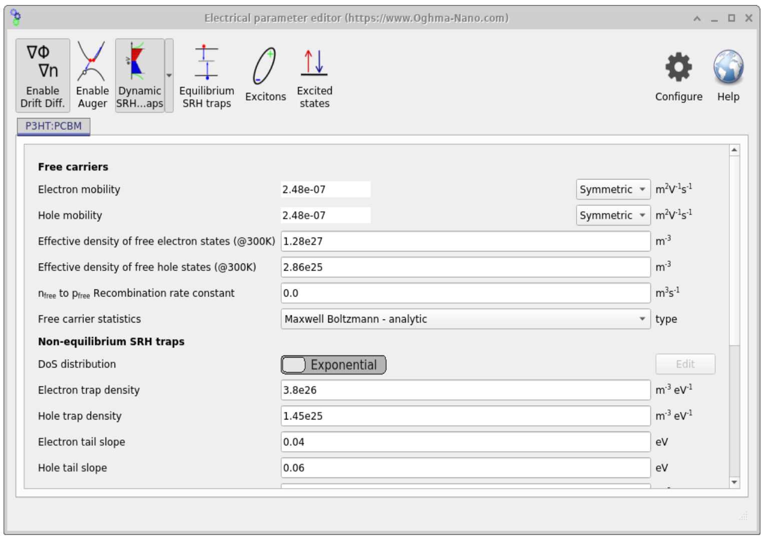
- Name: The name of the contact, this can be any English word. It has no physical meaning.
- Top/Bottom: Sets if the contact is on the top, bottom or in 2D simulation left and right of the device are also valid.
- Applied voltage: Sets the applied voltage on the contact. You first have to select what 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 wihtin the layer editor it will apear within the electrical paramter editor. The toolbar at the top of the window allows you to turn off and on various electrical mechanisms including:

- Drift diffusion: This enabled 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 9.6 for more information.
- Dynamic SRH traps: This is used to turn on and off dynamic SRH traps. See section 9.4 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 9.4 for more information.
- Excitons: This enables the exciton diffusion equation to be solved along with the electrical equations. See section 13.2 for more information.
- Excitons: This enables singlet and triplet states to be modelled.



Free carriers			
Electron mobility	2.48e-07	Symmetric	$m^2V^{-1}s^{-1}$
Hole mobility	2.48e-07	Symmetric	$m^2V^{-1}s^{-1}$
Effective density of free electron states (@300K)	1.28e27		m^{-3}
Effective density of free hole states (@300K)	2.86e25		m^{-3}
n_{free} to p_{free} Recombination rate constant	0.0		m^3s^{-1}
Free carrier statistics	Maxwell Boltzmann - analytic		type
Non-equilibrium SRH traps			
DoS distribution	Exponential	Edit	
Electron trap density	3.8e26		m^{-3} eV^{-1}
Hole trap density	1.45e25		m^{-3} eV^{-1}
Electron tail slope	0.04		eV
Hole tail slope	0.06		eV

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 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:

$$R(x) = kn(x)p(x) \tag{3.2}$$

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 (99.999999999%) (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 piggidly 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 suppler 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 16 (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
  - jv: 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.

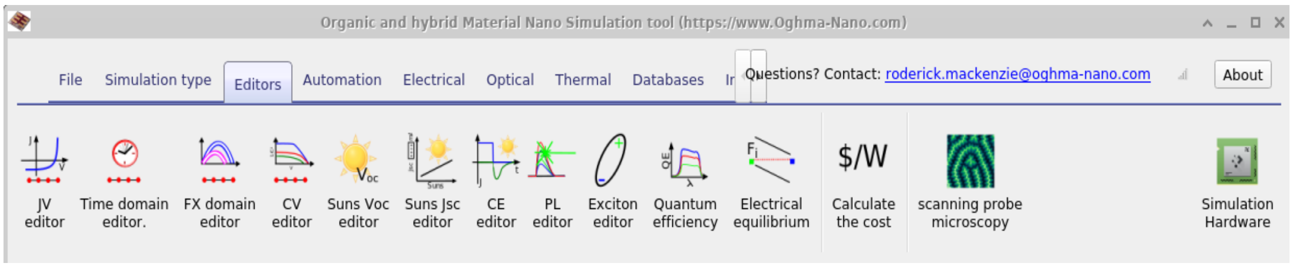
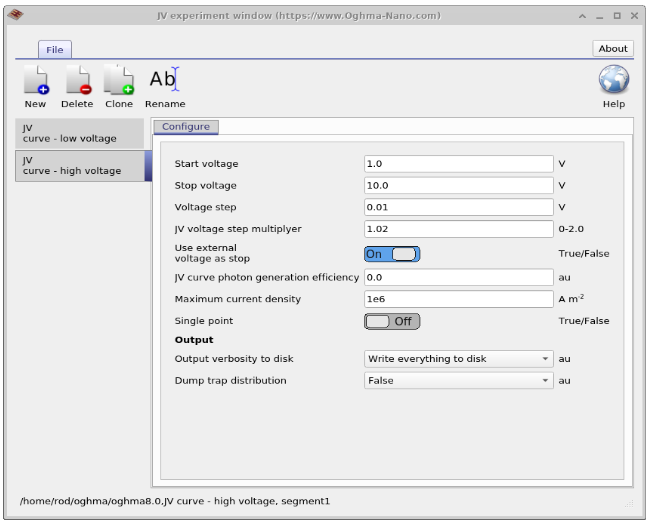


Figure 4.1: Simulation editors use this toolbar to edit the various simulation conditions your device will experience.

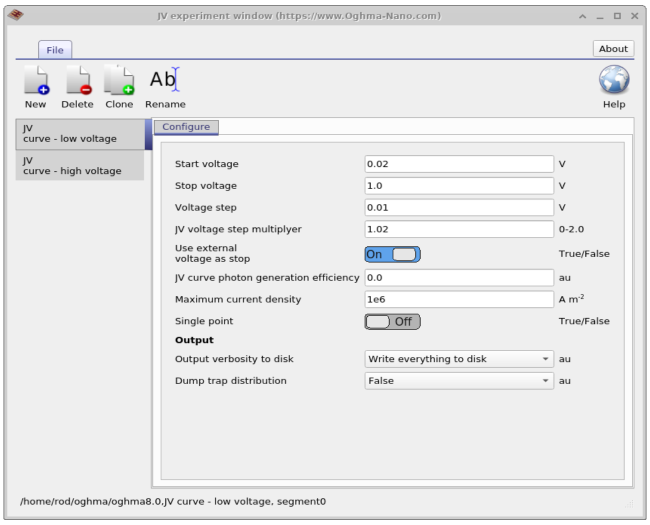
In the simulation editors ribbon (see Figure 4.1) you can see icons that represent each plugin, these are the simulation editors. By clicking on an icon in this ribbon you will be able to edit how the plugin performs the various simulations. For example in the JV simulation editor one can change the 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.



Start voltage	1.0	V
Stop voltage	10.0	V
Voltage step	0.01	V
JV voltage step multiplyer	1.02	0-2.0
Use external voltage as stop	On	True/False
JV curve photon generation efficiency	0.0	au
Maximum current density	1e6	A m^{-2}
Single point	Off	True/False
Output		
Output verbosity to disk	Write everything to disk	au
Dump trap distribution	False	au

/home/rod/oghma/oghma8.0,JV curve - high voltage, segment1

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



Start voltage	0.02	V
Stop voltage	1.0	V
Voltage step	0.01	V
JV voltage step multiplyer	1.02	0-2.0
Use external voltage as stop	On	True/False
JV curve photon generation efficiency	0.0	au
Maximum current density	1e6	A m^{-2}
Single point	Off	True/False
Output		
Output verbosity to disk	Write everything to disk	au
Dump trap distribution	False	au

/home/rod/oghma/oghma8.0,JV curve - low voltage, segment0

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.

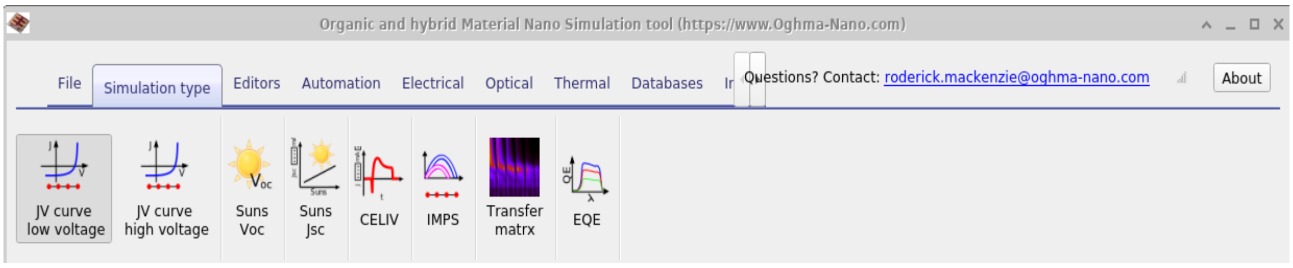


Figure 4.4: Selecting a simulation mode, in this case the *JV curve low voltage* has been selected so that when the user presses play that simulation mode will be run.

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.

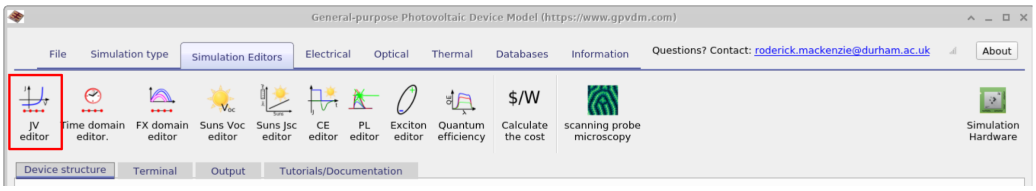
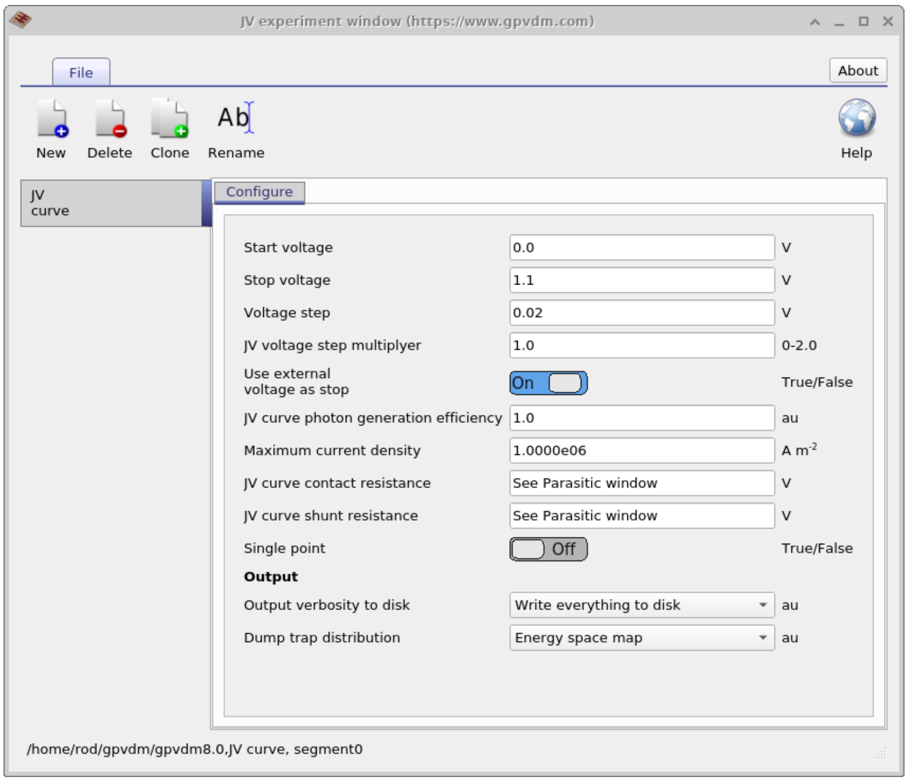


Figure 4.5: Opening the JV editor from the simulation editor ribbon.

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 multiplayer* 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 multiplyer 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 strugle to converge.



Start voltage	0.0	V
Stop voltage	1.1	V
Voltage step	0.02	V
JV voltage step multiplyer	1.0	0-2.0
Use external voltage as stop	On	True/False
JV curve photon generation efficiency	1.0	au
Maximum current density	1.0000e06	A m^{-2}
JV curve contact resistance	See Parasitic window	V
JV curve shunt resistance	See Parasitic window	V
Single point	Off	True/False
Output		
Output verbosity to disk	Write everything to disk	au
Dump trap distribution	Energy space map	au

/home/rod/gpvdm/gpvdm8.0.JV curve, segment0

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 10.1. As well as these files, by default OghmaNano will also write all internal simulation parameters to disk in the snapshots 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 19.1. 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
jv.dat	Current voltage curve
charge.dat	voltage charge density
k.csv	Recombination constant k
sim_info.dat	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}			
voc_R	voc_R	Recombination rate at P_{max}			
jv_{voc}	jv_voc				
jv_{pmax}	jv_pmax				
voc_{nt}	voc_nt	Trapped electron carrier densiyt at V_oc			
voc_{pt}	voc_pt	Trapped hole carrier density at V_oc			
voc_{nf}	voc_nf	Free electron carrier densiyt at V_oc			
voc_{pf}	voc_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			
voc_{np}	voc_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}	mu_jsc	Avg. mobility at J_{sc}	$m^2V^{-1}s^{-1}$		
μ_{jsc}^{geom}	mu_geom_jsc	Geom. avg. mobility @ J_{sc}	$m^2V^{-1}s^{-1}$		
$\mu_{jsc}^{geom\_micro}$	$mu\_geom\_micro\_jsc$	Geom. avg. mobility @ J_{sc}	$m^2V^{-1}s^{-1}$		
μ_{voc}	mu_voc	Average mobility @ V_{oc}	$m^2V^{-1}s^{-1}$		
μ_{voc}^{geom}	mu_geom_voc	Geom. avg. mobility @ V_{oc}	$m^2V^{-1}s^{-1}$	$\sqrt{\langle\mu_e\rangle\langle\mu_h\rangle}$	
$\mu_{voc}^{geom\_avg}$	$mu\_geom\_micro\_voc$	Geom. avg. mobility @ V_{oc}	$m^2V^{-1}s^{-1}$	$\langle\sqrt{mu_e\mu_h}\rangle$	
μ_{pmax}^{e}	mu_e_pmax	Avg. electron mobility @ P_{max}	$m^2V^{-1}s^{-1}$		
μ_{pmax}^{h}	mu_h_pmax	Avg. hole mobility @ P_{max}	$m^2V^{-1}s^{-1}$		
μ_{pmax}^{geom}	mu_geom_pmax	Geom. avg. mobility @ P_{max}	$m^2V^{-1}s^{-1}$	$\sqrt{\langle\mu_e\rangle\langle\mu_h\rangle}$	
$\mu_{pmax}^{geom\_micro}$	$mu\_geom\_micro\_pmax$	Geom. avg. mobility @ P_{max}	$m^2V^{-1}s^{-1}$	$\langle\sqrt{mu_e\mu_h}\rangle$	
μ^{pmax}	mu_pmax	Avg. mobility @ P_{max}	$m^2V^{-1}s^{-1}$		

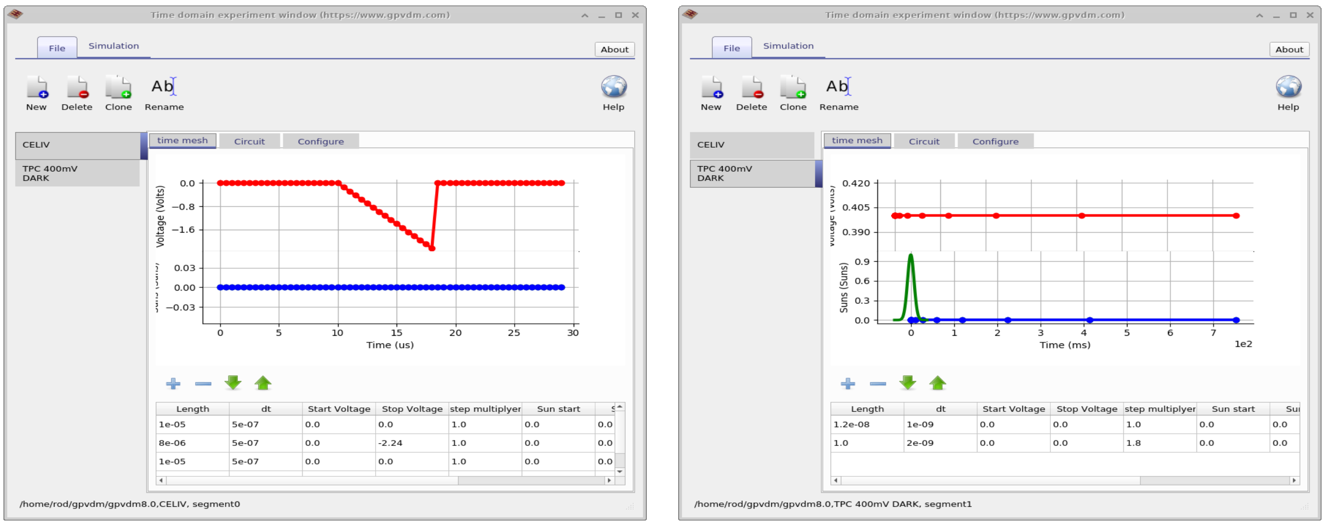
Symbol	JSON token	Meaning	Units	Equ.	Ref
τ_{voc}	*tau_voc*	Recom. time at V_{oc}	s	$R = (n - n0)/\tau$	[7]
τ_{pmax}	*tau_pmax*	Recom. time at P_{max}	s	$R = (n - n0)/\tau$	[7]
τ_{voc}^{all}	*tau_all_voc*	Recomb. time at V_{oc}	s	$R = (n)/\tau$	[7]
τ_{pmax}^{all}	*tau_all_pmax*	Recomb. time at P_{max}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



Length	dt	Start Voltage	Stop Voltage	step multiplyer	Sun start	S
1e-05	5e-07	0.0	0.0	1.0	0.0	0.0
8e-06	5e-07	0.0	-2.24	1.0	0.0	0.0
1e-05	5e-07	0.0	0.0	1.0	0.0	0.0

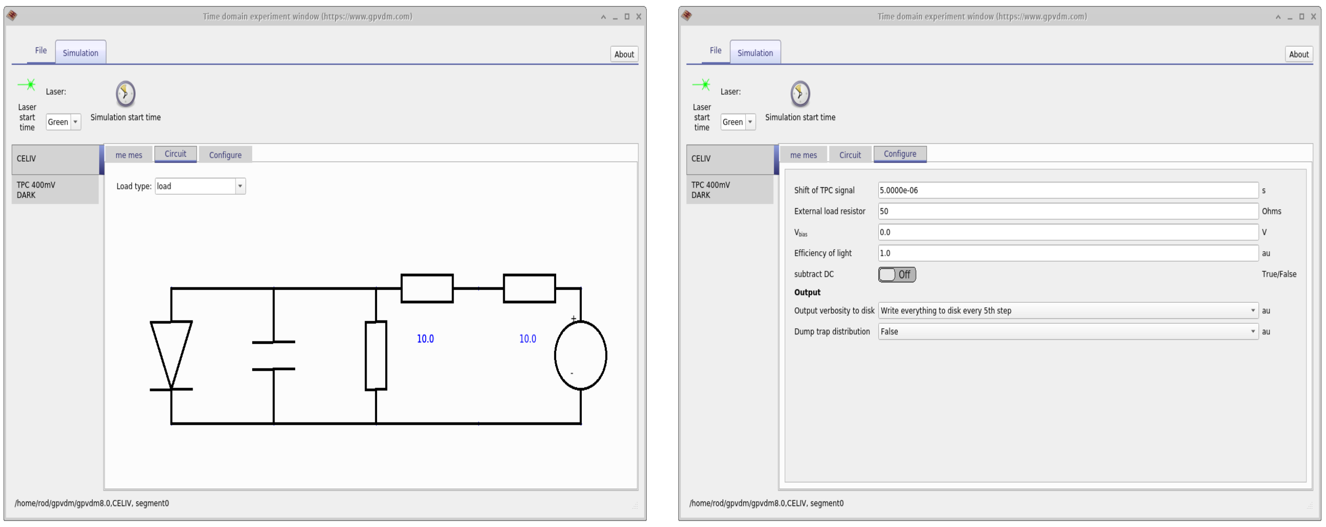
/home/rod/gpvdm/gpvdm8.0,CELIV, segment0

Length	dt	Start Voltage	Stop Voltage	step multiplyer	Sun start	Su
1.2e-08	1e-09	0.0	0.0	1.0	0.0	0.0
1.0	2e-09	0.0	0.0	1.8	0.0	0.0

/home/rod/gpvdm/gpvdm8.0,TPC 400mV DARK, segment1

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.



Load type: load

Shift of TPC signal 5.0000e-06 s
External load resistor 50 Ohms
V_{bias} 0.0 V
Efficiency of light 1.0 au
subtract DC Off True/False

Output

Output verbosity to disk Write everything to disk every 5th step au
Dump trap distribution False au

/home/rod/gpvdm/gpvdm8.0,CELIV, segment0

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

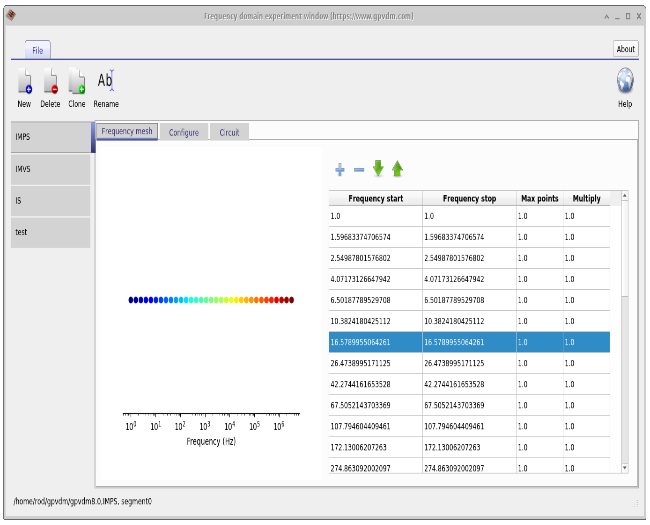
4.3 Frequency domain editor

Related YouTube videos:

Simulating impedance spectroscopy (IS) in solar cells.

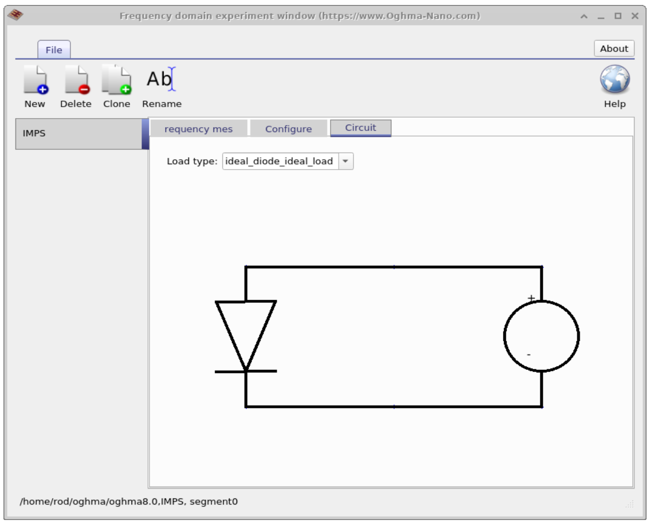
4.3.1 Overview

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.



Frequency start	Frequency stop	Max points	Multiply
1.0	1.0	1.0	1.0
1.59683374706574	1.59683374706574	1.0	1.0
2.54987801576802	2.54987801576802	1.0	1.0
4.07173126647942	4.07173126647942	1.0	1.0
6.50187789529708	6.50187789529708	1.0	1.0
10.3824180425112	10.3824180425112	1.0	1.0
16.5789955064261	16.5789955064261	1.0	1.0
26.4738995171125	26.4738995171125	1.0	1.0
42.2744161653528	42.2744161653528	1.0	1.0
67.5052143703369	67.5052143703369	1.0	1.0
107.794604409461	107.794604409461	1.0	1.0
172.13006207263	172.13006207263	1.0	1.0
274.863092002097	274.863092002097	1.0	1.0

Figure 4.9: The frequency domain editor window



Frequency domain experiment window (https://www.Oghma-Nano.com)
File
About
New
Delete
Clone
Rename
Help
IMPS
requency mes
Configure
Circuit
+
-
/home/rod/oghma/oghma8.0,IMPS, segment0

Load type: ideal_diode_ideal_load

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.

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

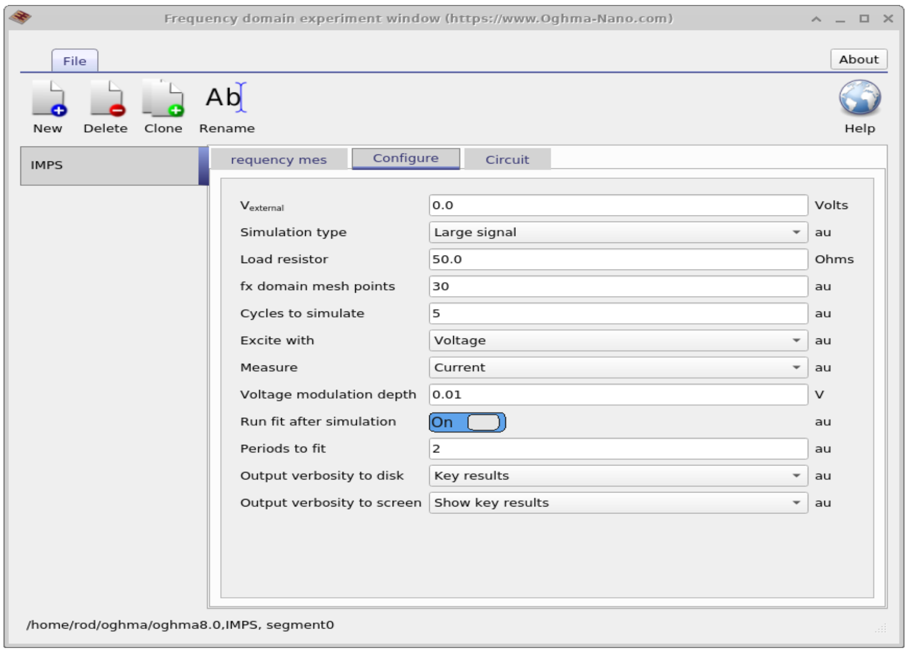
4.3.2 Inputs

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. $\lvert i(fx)\rvert$
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



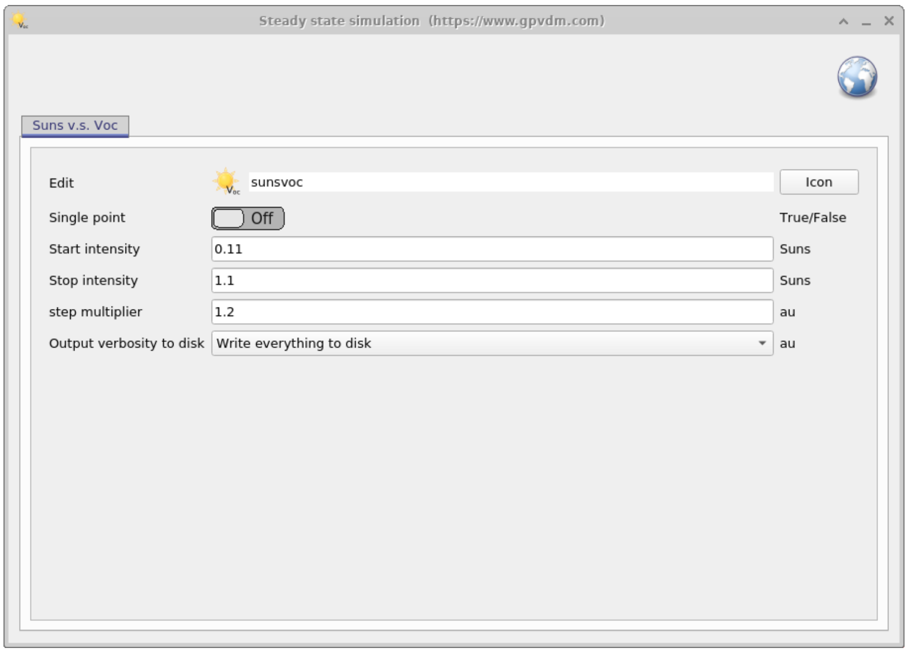
$V_{external}$	0.0	Volts
Simulation type	Large signal	au
Load resistor	50.0	Ohms
fx domain mesh points	30	au
Cycles to simulate	5	au
Excite with	Voltage	au
Measure	Current	au
Voltage modulation depth	0.01	V
Run fit after simulation	On	au
Periods to fit	2	au
Output verbosity to disk	Key results	au
Output verbosity to screen	Show key results	au

/home/rod/oghma/oghma8.0,IMPS, segment0

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*.



Edit	sunsvoc	
Single point	Off	True/False
Start intensity	0.11	Suns
Stop intensity	1.1	Suns
step multiplier	1.2	au
Output verbosity to disk	Write everything to disk	au

Figure 4.12: The suns-voc editor window

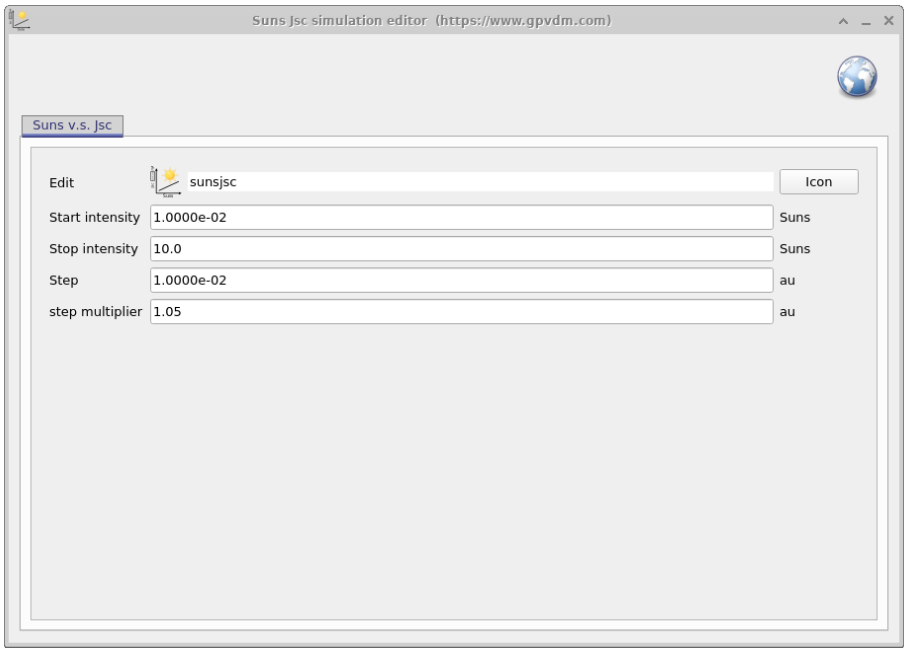
4.4.1 Outputs

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.



Edit	sunsjsc	
Start intensity	1.0000e-02	Suns
Stop intensity	10.0	Suns
Step	1.0000e-02	au
step multiplier	1.05	au

Figure 4.13: The JV curve editor window

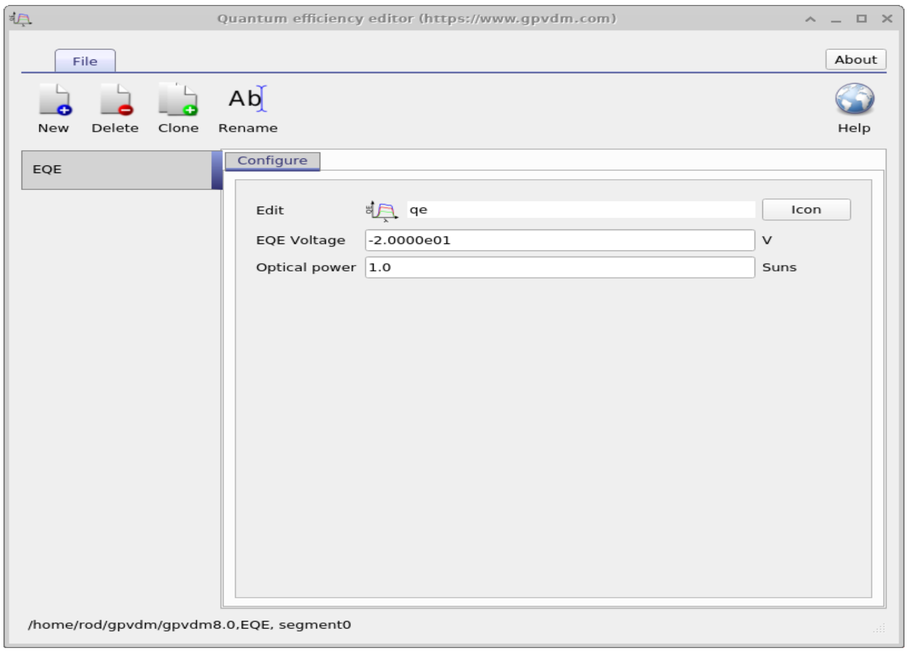
4.5.1 Outputs

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.



Edit qe
EQE Voltage -2.0000e01 V
Optical power 1.0 Suns

/home/rod/gpvdm/gpvdm8.0,EQE, segment0

Figure 4.14: The quantum efficiency editor window

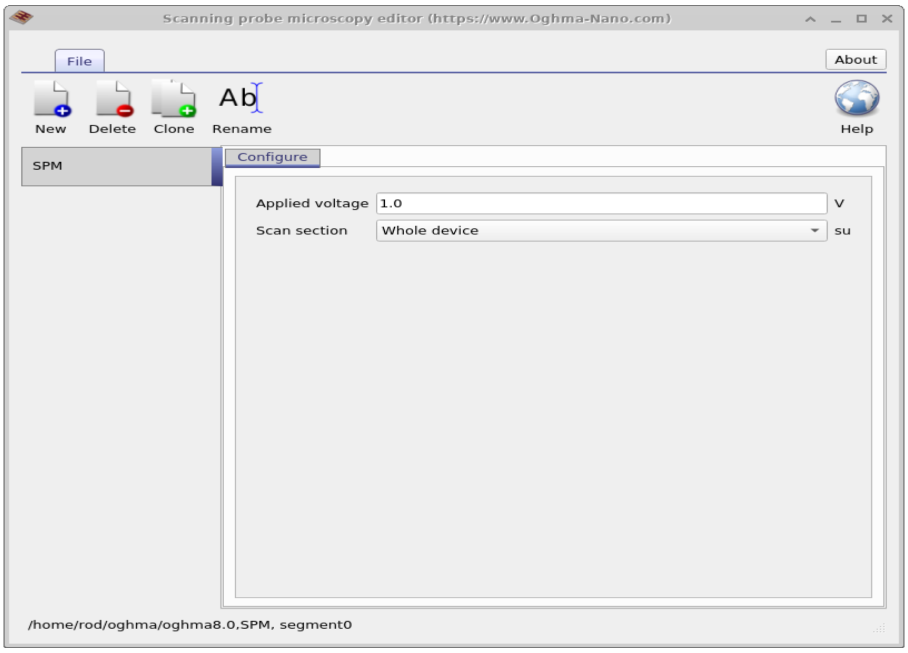
4.6.1 Outputs

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 12.


Scanning probe microscopy editor (https://www.Oghma-Nano.com)
File
About
New
Delete
Clone
Rename
Help
SPM
Configure

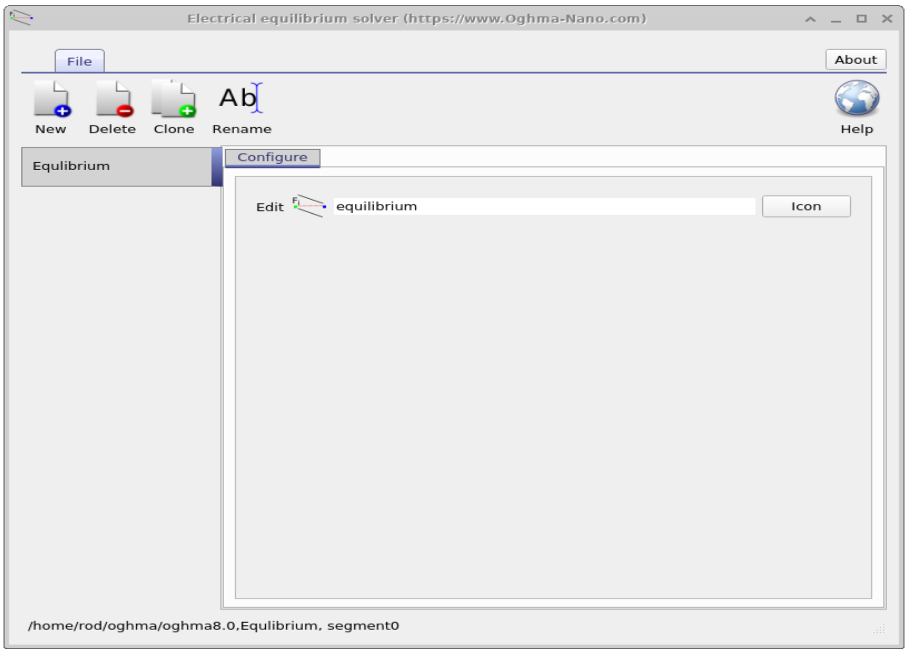
Applied voltage 1.0 V
Scan section Whole device su

/home/rod/oghma/oghma8.0,SPM, segment0

Figure 4.15: The scanning probe microscopy editor

4.8 Electrical equilibrium editor

Sometimes when studding 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).



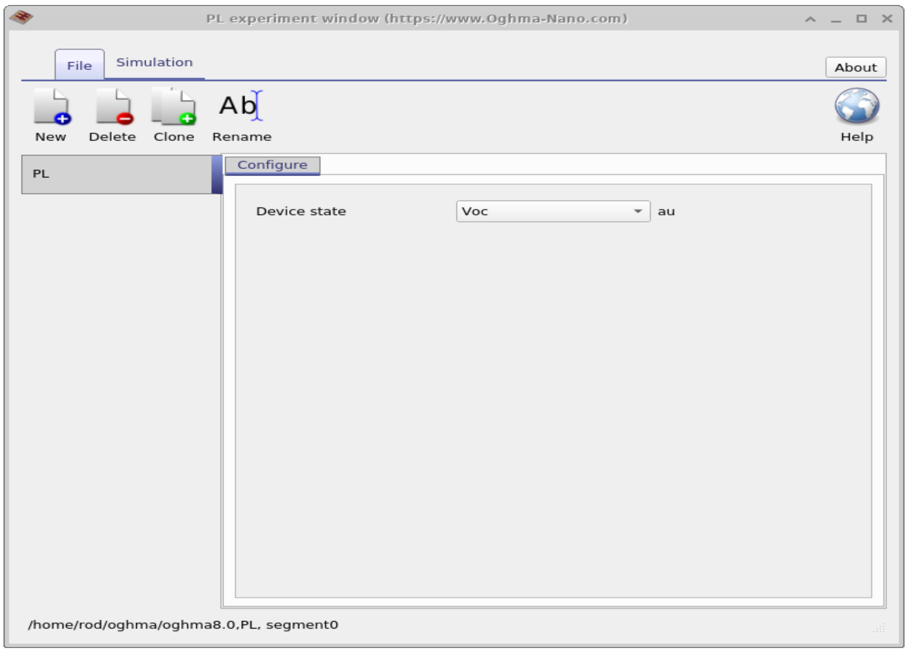
Edit equilibrium

/home/rod/oghma/oghma8.0,Equlibrium, segment0

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.



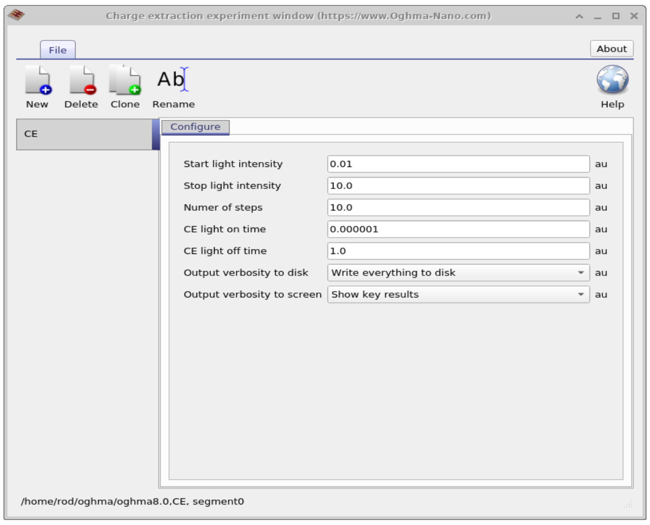
Device state Voc au

/home/rod/oghma/oghma8.0,PL, segment0

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.


Charge extraction experiment window (https://www.Oghma-Nano.com)
File
About
New
Delete
Clone
Rename
Help
CE
Configure

Start light intensity	0.01	au
Stop light intensity	10.0	au
Numer of steps	10.0	au
CE light on time	0.000001	au
CE light off time	1.0	au
Output verbosity to disk	Write everything to disk	au
Output verbosity to screen	Show key results	au

/home/rod/oghma/oghma8.0,CE, segment0

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.

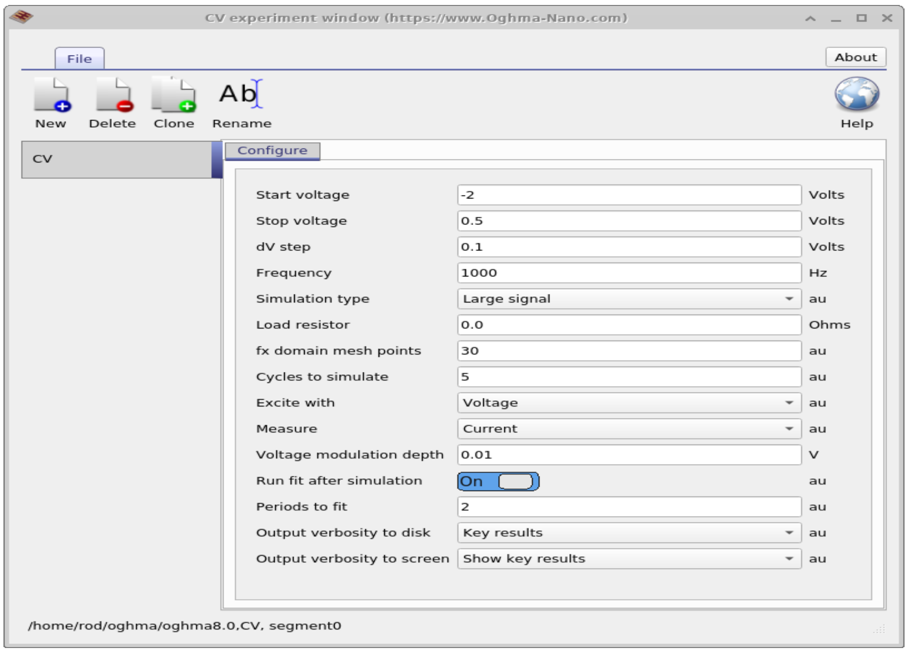
4.10.1 Outputs

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



Start voltage	-2	Volts
Stop voltage	0.5	Volts
dV step	0.1	Volts
Frequency	1000	Hz
Simulation type	Large signal	au
Load resistor	0.0	Ohms
fx domain mesh points	30	au
Cycles to simulate	5	au
Excite with	Voltage	au
Measure	Current	au
Voltage modulation depth	0.01	V
Run fit after simulation	On	au
Periods to fit	2	au
Output verbosity to disk	Key results	au
Output verbosity to screen	Show key results	au

/home/rod/oghma/oghma8.0.CV, segment0

Figure 4.19: The capacitance voltage editor

4.11.1 Outputs

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

Table 4.8: Files produced by the CV simulation

4.12 Hardware editor

All computer programs including OghmaNano run on physical computing hardware. There are may combinations of hardware that can be in any computer, some computes have a large number of CPU cores while others only have one. Likewise computers come with differing amounts of memory, hard disk space and GPUs. To help the user get the best out of OghmaNano, there is a hardware editor where the user can configure how OghmaNano behaves on any given computer. This can be accessed through the simulation tab window see Figure 4.20.

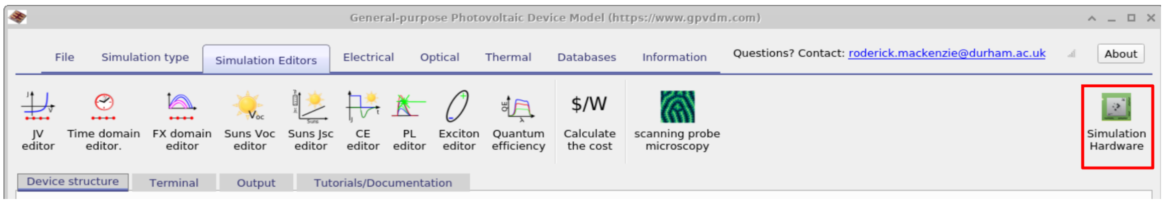
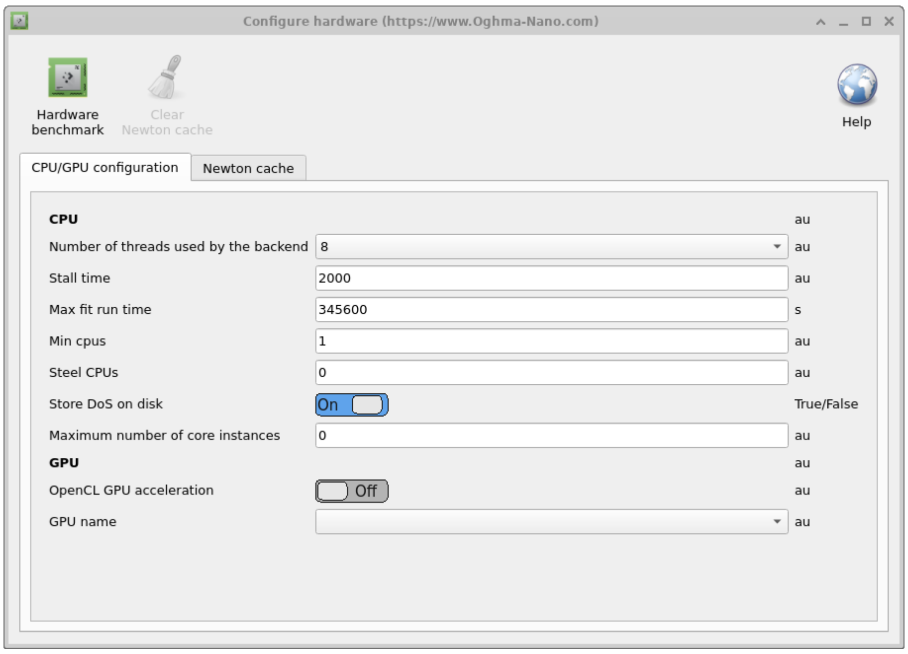


Figure 4.20: Opening the hardware editor.

If you click on this it will bring up the hardware editor window which can be seen in Figure 4.21



CPU		au
Number of threads used by the backend	8	au
Stall time	2000	au
Max fit run time	345600	s
Min cpus	1	au
Steel CPUs	0	au
Store DoS on disk	On	True/False
Maximum number of core instances	0	au
GPU		au
OpenCL GPU acceleration	Off	au
GPU name		au

Figure 4.21: The hardware editor window

The hardware window is comprised of various tabs which enable the user to edit the configuration and also benchmark your device.

4.12.1 CPU/GPU configuration tab

This tab is used to configure how OghmaNano interacts with the GPU and CPU it is described in the table below. As described in other parts of this manual in detail there are two parts to OghmaNano there is oghma_core.exe which is the computational back end and there is oghma_gui.exe which is the graphical user interface, how both these parts of the model behave can be fine tuned here.

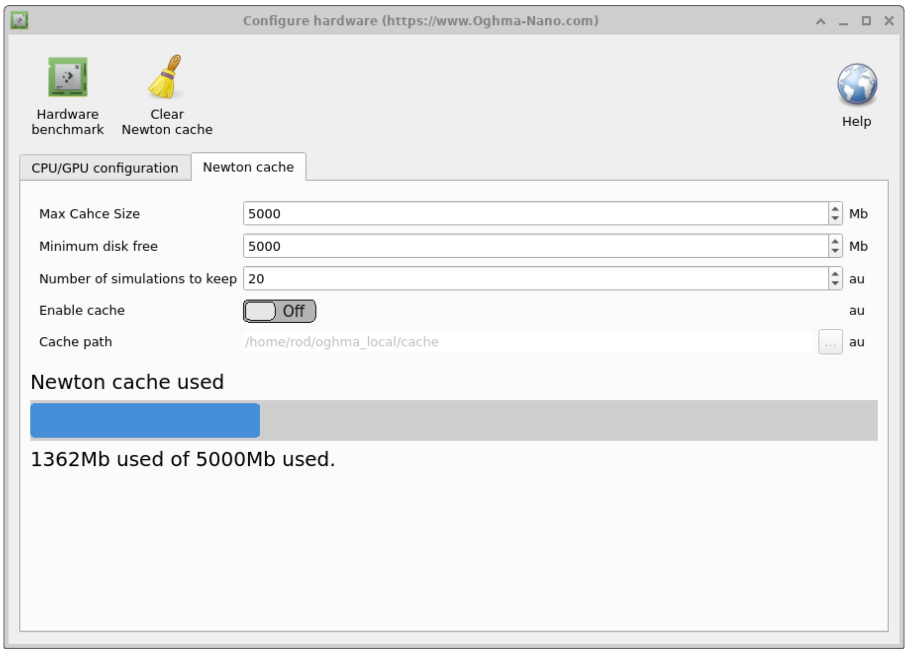
- *Number of threads used by the backend:* This is the maximum number of threads OghmaNano oghma_core.exe can use. This dictates; the number of simultaneous fits that can be run; the maximum number of optimization simulations that can be run at the same time; the maximum number of threads that are used for FDTD simulations; maximum number of of DoS cache files can generated at the same time; number of frequency domain points that can be run at the same time.

- *Maximum number of core instances:* This sets the maximum number of oghma_core.exe instances that can be started by the GUI. If one is running a parameter scan then this will control the maximum number of simultaneous simulations that can be performed at the same time. If the values of *Number of threads used by the backend* has been set to 4 and one is performing an FDTD simulation, then one sets *Maximum number of core instances* to 8, then the GUI will spawn 8 instances of oghma_core.exe each using 4 threads, thus 32 CPU cores will be needed.
- *Stall time:* Sometimes when running OghmaNano on a supercomputer unattended it can stop running, possibly because of an IO error or network error. This option can be used to set the maximum length of a single simulation. By single simulation I mean a single JV curve, single time domain simulation or single frequency domain simulation, but not a whole fit which will involve running thousands if individual simulations.). So with a value of 2000 seconds the solver will exit, if for example a single JV simulation takes longer than 2000 seconds. In reality any individual simulation should take only a few seconds, so this option acts as a hard backstop if something has gone very wrong.
- *Max fit run time:* This is the maximum time oghma_core.exe can reside in memory. If any simulation or fit takes longer than this value it will be terminated, again this is a backstop to prevent simulations running forever. The default value is 4 days.
- *Steel CPUs:* Sometimes when running OghmaNano on a shared PC one will set a simulation running when another user is using a significant number of cores. After a while the other user's simulations will finish running leaving the computer with idling CPUs. If this option is set to *True*, then OghmaNano will monitor the number of free CPUs and if more become available it will use them.
- *Min CPUs:* Used with the option above *Steel CPUs* to set the minimum number of CPUs that will be used.
- *Store DoS on disk:* OghmaNano stores lookup tables on disk to speed up simulations, if this option is set to false these lookup tables will not be stored.
- *OpenCL GPU acceleration:* This enables or disables GPU acceleration, this is used mainly during the FDTD simulations.
- *GPU name:* Selects the GPU to use.

4.12.2 Newton cache

When running simulations with a significant number of ODEs, such as 1D devices with a lot of trap states and a lot of spatial points, or when running 2D OFET simulations each voltage step can take a while to compute. This is because the solver must solve each voltage step using Newton's method until it converges. For each each solver step the Jacobian must be built, the matrix inverted multiplied by the residuals and updates to all solver variables calculated. This can take a significant amount of time per step (2000ms). An approach to side step this approach is to store previously calculated answers on disk and then when the user asks the solver to calculated an already calculated problem the answer can be recalled rather than recalculated. This is very useful in OLED design where one is trying to optimise the optical structure of the device but leaves the electrical structure unchanged. One can run new optical simulations with already pre-calcualted electrical solutions. Configuration options are displayed in the table below.

There is an overhead to using the Newton Cache, so I would only recommend it when solving the electrical problem is very slow indeed. Technically the Newton cache works by taking the MD5 sum of the Fermi-levels and the potentials to generate a hash of the electrical problem. This is then compared to what exists on disk. If a precalculated answer is found, the Fermi-levels/potentials are updated to the values found on disk. The cache is stored in oghma_local cache, each pre solvedsolution is stored as a new binary file. Each simulation run generates an index file where all MD5 sums from that simulation are stored. Once the cache becomes full OghmaNano deletes simulation results in batches based on the index files.



Max Cahce Size 5000 Mb
Minimum disk free 5000 Mb
Number of simulations to keep 20 au
Enable cache Off au
Cache path /home/rod/oghma_local/cache au

Newton cache used

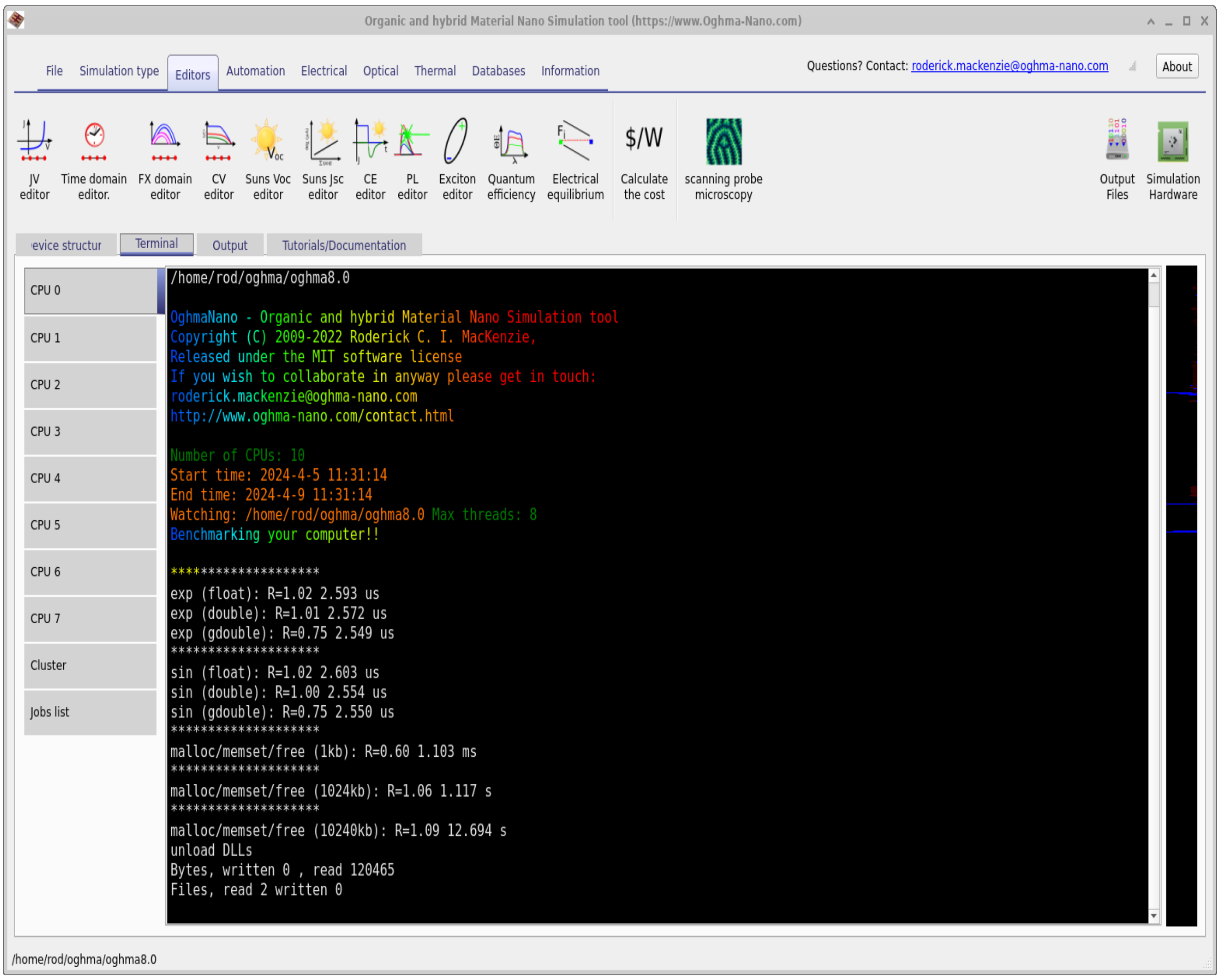
1362Mb used of 5000Mb used.

Figure 4.22: The Newton cache editor

- *Maximum cache size:* Sets the maximum size of the cache in Mb. I would recommend around 1Gb.
- *Minimum disk free:* Sets the minimum amount of disk space needed to use the cache, this option is designe dto prevent the cache filling the disk I would set it to around 5Gb.
- *Number of simulations to keep:* This will set the maximum number of simulation runs to keep, I would set it to between 20 and 100.
- *Enable cache:* This enables or disables the Newton Cache, the default and recommended option is False.

4.12.3 Hardware benchmark

In the top left of hardware window 4.22 there is a button called *Hardware benchmark*. If this is clicked then OghmaNano will benchmark your hardware, the result of such a benchmark can be seen in 4.23. This runs benchmarks your CPUs ability to calculate *sin*,*exp* and allocate/deallocate memory in blocks. It displays how long it took to do a few thousand operations as well as an *R* (aka Roderick) value. This is defined as R=*Time taken to do the calculation on your PC/The time take to do the calculation on my PC*. Thus smaller values mean your PC is faster than mine. My PC is a Intel(R) Core(TM) i7-4900MQ CPU @ 2.80GHz in a 2017 Lenovo thinkpad. So most modern computers should be faster. If you have good CPU performance but your simulations are running slower than my YouTube videos then this is invariably due to bad IO speed, caused by virus killers, storing the simulations on OneDrive, using networked drives, using slow USB storage etc.



```
/home/rod/oghma/oghma8.0

OghmaNano - Organic and hybrid Material Nano Simulation tool
Copyright (C) 2009-2022 Roderick C. I. MacKenzie,
Released under the MIT software license
If you wish to collaborate in anyway please get in touch:
roderick.mackenzie@oghma-nano.com
http://www.oghma-nano.com/contact.html

Number of CPUs: 10
Start time: 2024-4-5 11:31:14
End time: 2024-4-9 11:31:14
Watching: /home/rod/oghma/oghma8.0 Max threads: 8
Benchmarking your computer!!

********************
exp (float): R=1.02 2.593 us
exp (double): R=1.01 2.572 us
exp (gdouble): R=0.75 2.549 us
********************
sin (float): R=1.02 2.603 us
sin (double): R=1.00 2.554 us
sin (gdouble): R=0.75 2.550 us
********************
malloc/memset/free (1kb): R=0.60 1.103 ms
********************
malloc/memset/free (1024kb): R=1.06 1.117 s
********************
malloc/memset/free (10240kb): R=1.09 12.694 s
unload DLLs
Bytes, written 0 , read 120465
Files, read 2 written 0
```

Figure 4.23: Running a hardware benchmark

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 9.11 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 capter we will be looking at the (standard) top contact OFET (Figure ??), double click on this and save the new simulation to disk.

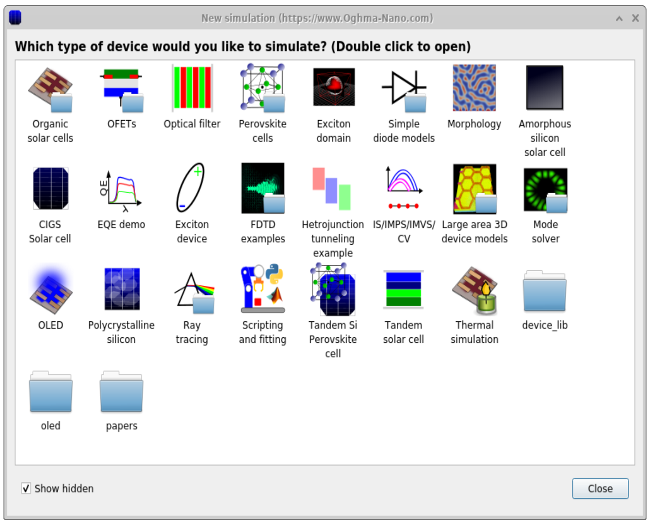


Figure 5.1: Setlect the OFET submenu to main a new OFET simulation

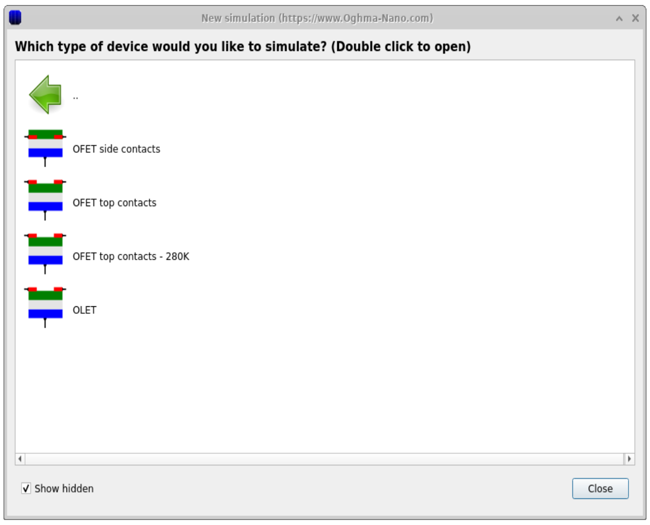
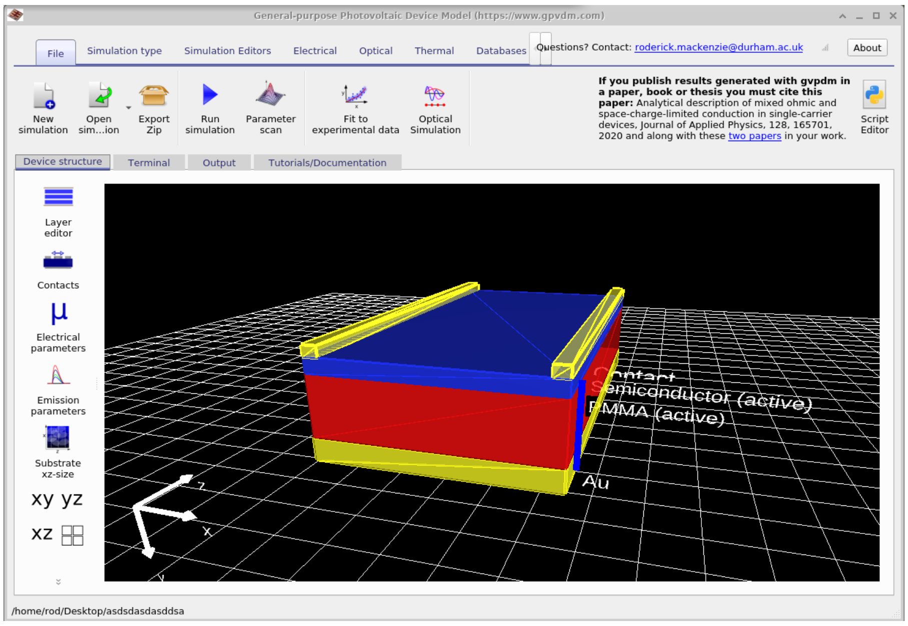

New simulation (https://www.Oghma-Nano.com)
Which type of device would you like to simulate? (Double click to open)
..
OFET side contacts
OFET top contacts
OFET top contacts - 280K
OLET
Show hidden
Close

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

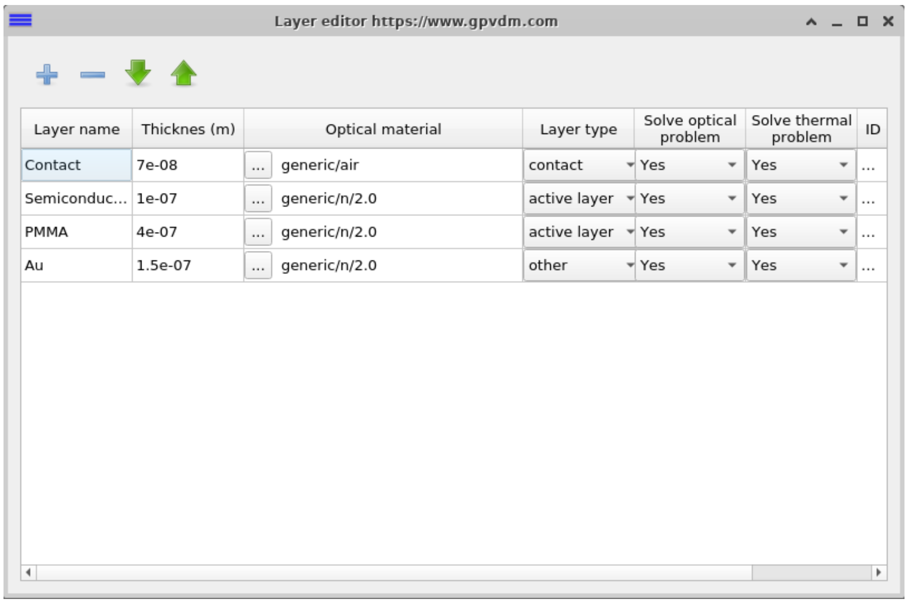
5.0.1 The anatomy of a 2D simulation



If you publish results generated with gvpdm in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

Figure 5.3: The default ofet simulation.

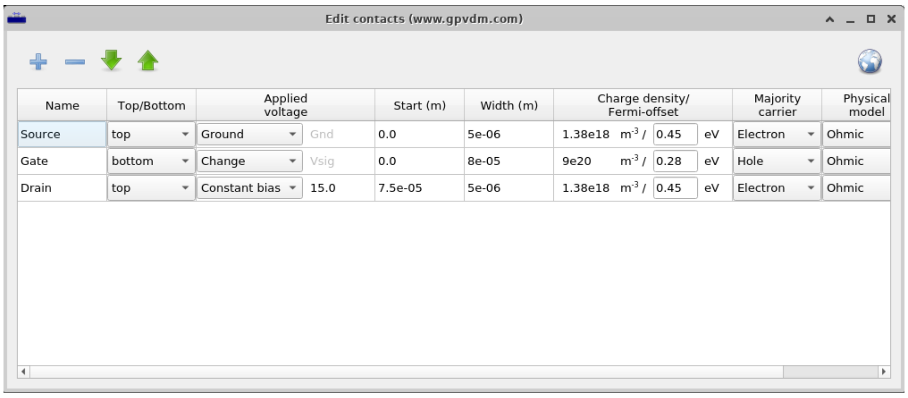
The OFET structure shown in Figure 5.3 has three contacts, a *gate*, *source* and a *drain*. The source and drain are shown on the top of the simulation as gold bars, a semiconductor layer is shown in blue and an insulating later shown in red. The *gate* contact is visible at the bottom of the structure. 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	Thicknes (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 5μm, the *drain* starts at 75 μm and extends to 5μm, while the gate starts at 0 m and extends to cover the entire width of the device which is 80 μm. 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.


Edit contacts (www.gpvdm.com)

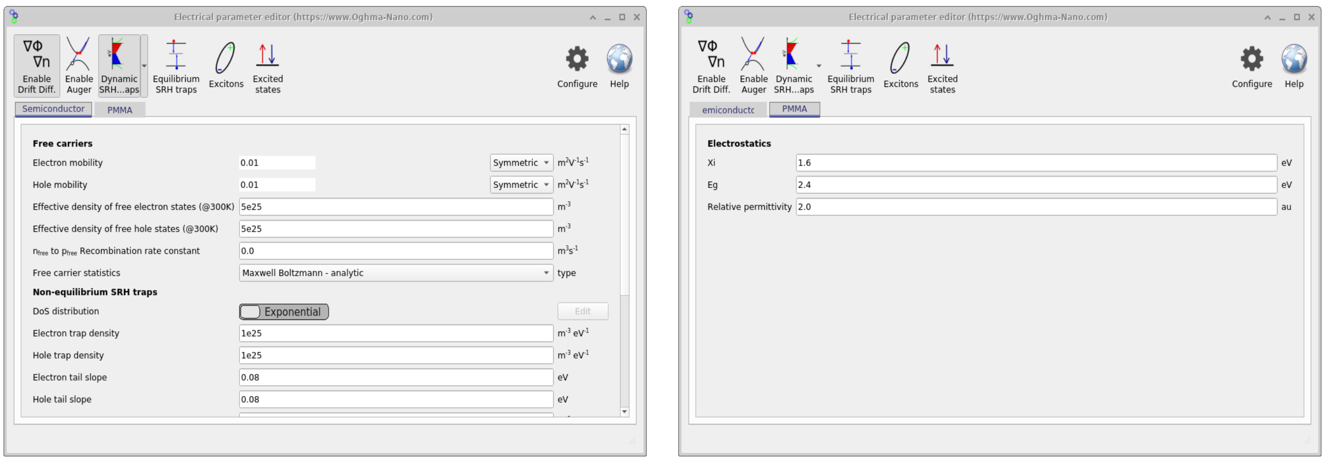
Name	Top/Bottom	Applied voltage		Start (m)	Width (m)	Charge density/ Fermi-offset		Majority carrier	Physical model
Source	top	Ground	Gnd	0.0	5e-06	1.38e18 m^{-3} /	0.45 eV	Electron	Ohmic
Gate	bottom	Change	Vsig	0.0	8e-05	9e20 m^{-3} /	0.28 eV	Hole	Ohmic
Drain	top	Constant bias	15.0	7.5e-05	5e-06	1.38e18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



Free carriers

Electron mobility	0.01	Symmetric	$m^2V^{-1}s^{-1}$
Hole mobility	0.01	Symmetric	$m^2V^{-1}s^{-1}$
Effective density of free electron states (@300K)	5e25		m^{-3}
Effective density of free hole states (@300K)	5e25		m^{-3}
n_{free} to p_{free} Recombination rate constant	0.0		m^3s^{-1}
Free carrier statistics	Maxwell Boltzmann - analytic		type

Non-equilibrium SRH traps

DoS distribution	Exponential	Edit	
Electron trap density	1e25		$m^{-3}\ eV^{-1}$
Hole trap density	1e25		$m^{-3}\ eV^{-1}$
Electron tail slope	0.08		eV
Hole tail slope	0.08		eV

Electrostatics

Xi	1.6	eV
Eg	2.4	eV
Relative permittivity	2.0	au

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.

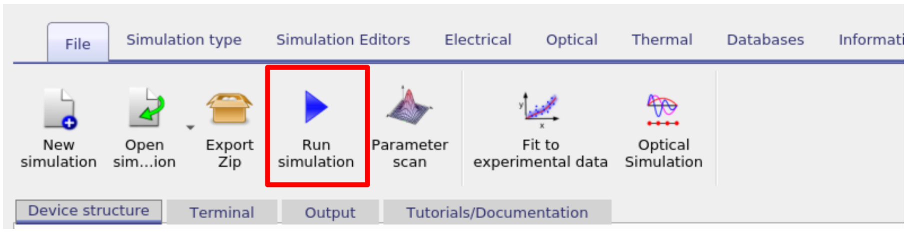


Figure 5.7: Running an OFET simulation

The simulation will take longer than it's 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
contact_iv0.dat	Current voltage current curve for contact 0
contact_iv1.dat	Current voltage current curve for contact 1
contact_iv2.dat	Current voltage current curve for contact 2
contact_jv0.dat	Current density voltage current curve for contact 0
contact_jv1.dat	Current density voltage current curve for contact 1
contact_jv2.dat	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.

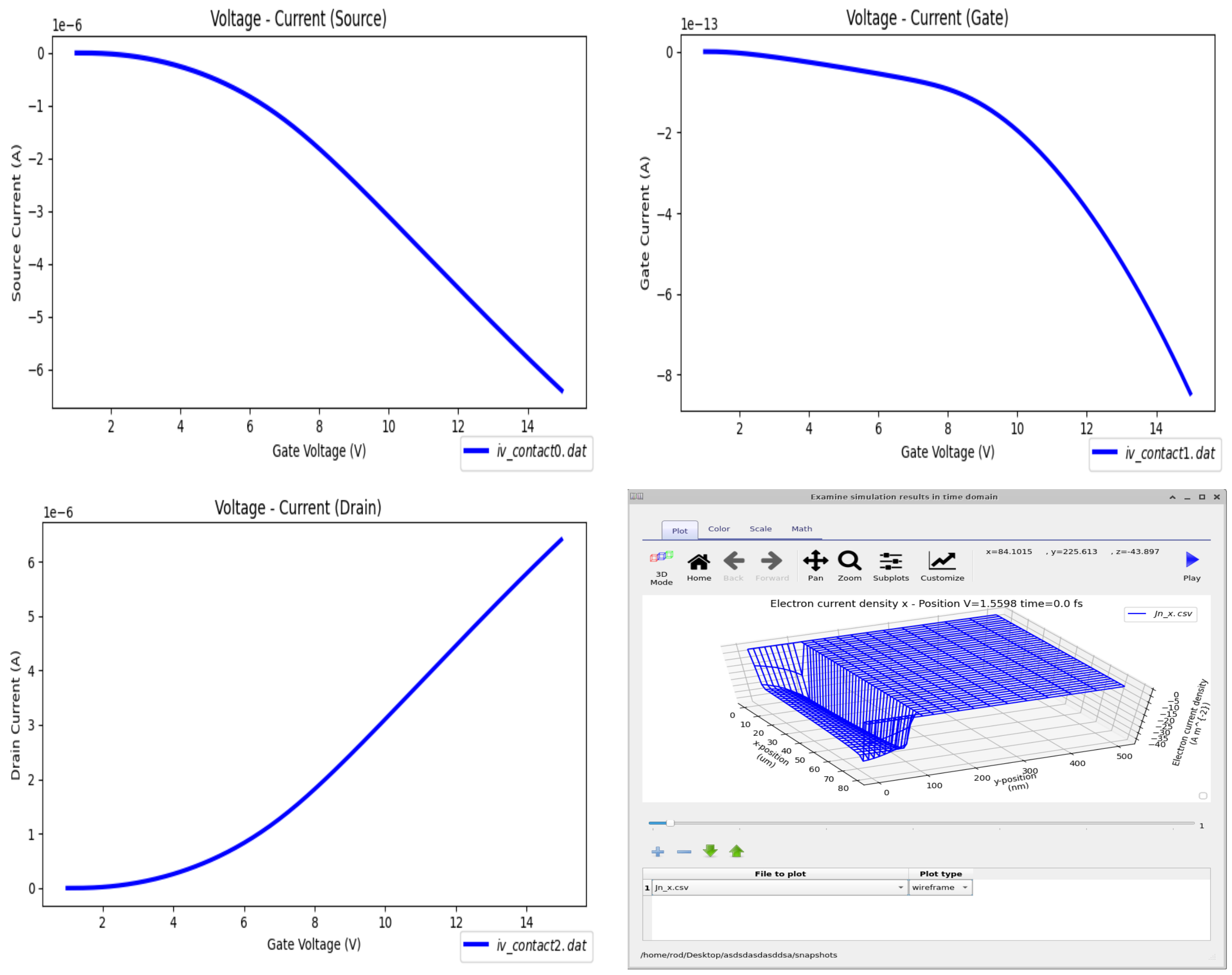


Figure 5.8: Results from a 2D OFET simulation the JV curves for each contact are shown along with a view of the the electron current density in the x direction (bottom right).

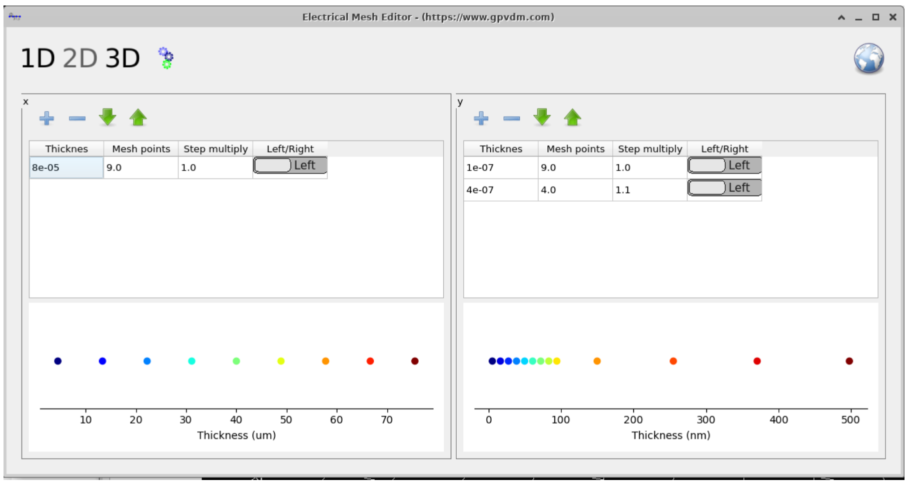
5.0.4 Meshing in 2D

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.



Thicknes	Mesh points	Step multiply	Left/Right
8e-05	9.0	1.0	Left

Thicknes	Mesh points	Step multiply	Left/Right
1e-07	9.0	1.0	Left
4e-07	4.0	1.1	Left

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 $1x10^{-12} - 1x10^{-15} m^2/(Vs)$ to limit current flow into the region. However, the value should not be set too low (see section 9.11.1) 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 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 9.11.1 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

Organic Light Emitting Diodes (OLEDs)

Simulating OLEDs with multiple emission layers

Simulating OLED structures using ray tracing and drift diffusion

7.1 Introduction

Organic Light Emitting Diodes (OLEDs) are a widely used technology with applications in displays and sensing. The devices operate by charge being injected through the contacts which results in electro luminescence. The following chapter describes how to simulate these devices using OghmaNano. There are various OLED examples which can be accessed from the OLED section of the *New simulation window*a (Figure 7.1).

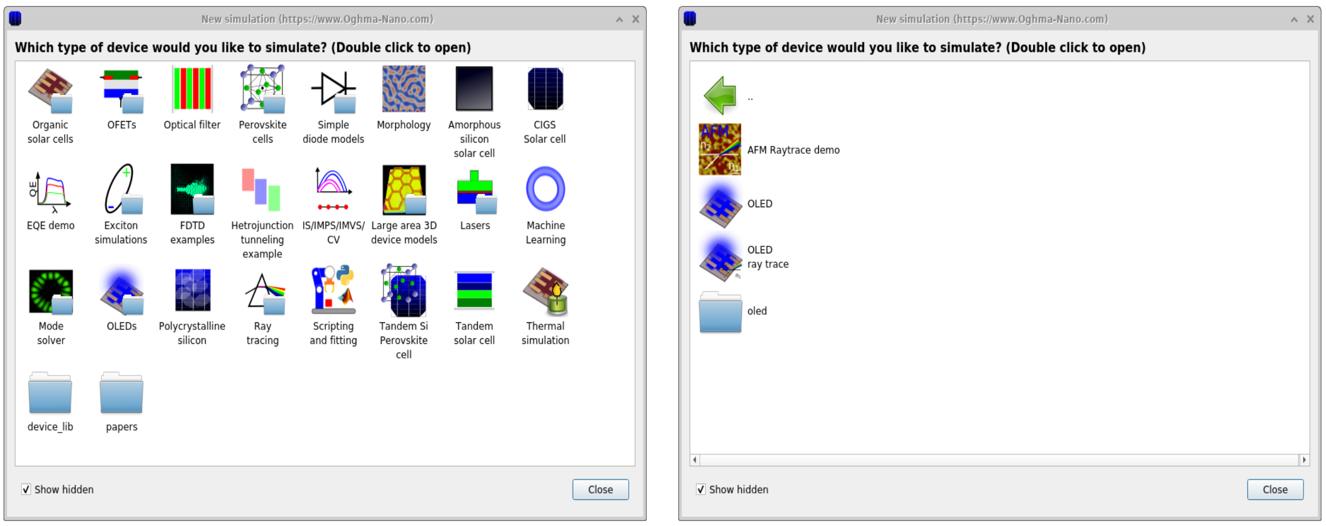
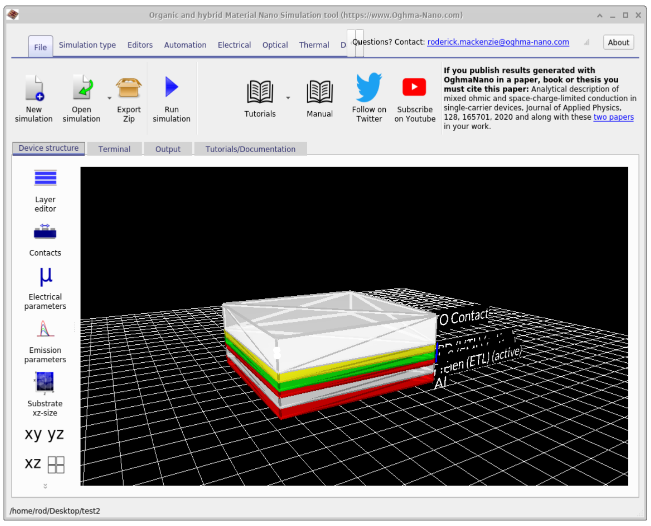


Figure 7.1: a) The new simulation window; b) OLED example simulations.

OLED simulations use a combination of the drift-diffusion models described elsewhere in this book combined with optical models to simulate the propagation of light out of the device. The *Transfer matrix method* and *ray tracing* can be used to model the escape of light from OLEDs within OghmaNano. The transfer matrix method is useful when one wishes understand how light behaves when propagating normal to the surface of smoothed layer structures such as in well defined evaporated OLEDs, while the ray tracing method is used when wishing to simulate light escaping from rough surfaces or structures with micro-lenses embedded.

7.2 Optical outcoupling

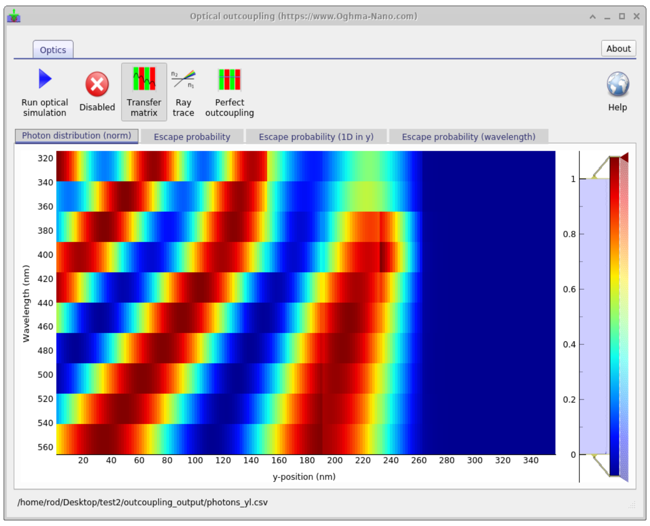
The most simple OLED simulation based on the transfer matrix method can be accessed by double clicking on the *OLED* example simulation (see Figure 7.1b). This will bring up the window shown in Figure 7.2a. In the *Optical* ribbon you will find a button called *Optical outcoupling*. This tool allows the user to select which if the *Transfer matrix method* or the *ray tracing* method is used. If this window is opened a window resembling that shown in Figure 7.2b will be displayed. By clicking *Run* in this window, the probability of emission as a function of position and wavelength will be calculated this will be displayed in the *Escape probability* tab. By clicking on *Ray trace* and then clicking on the run button again, the simulation will be repeated using the ray tracing method, the results form such a simulation are shown in Figure 7.3. Note that the outcoupling efficiency for ray tracing is lower than that predicted by the transfer matrix as the transfer matrix method assumes propagation normal to the interfaces while ray tracing allows rays to travel in all directions some of which will never leave the device.



Questions? Contact: roderick.mackenzie@oghma-nano.com

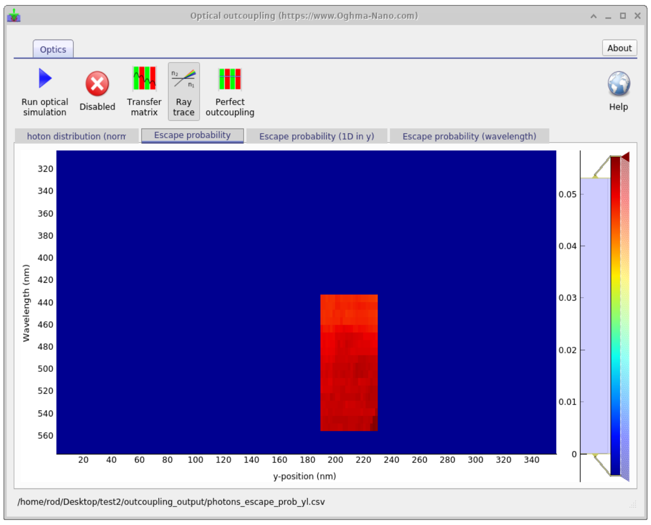
If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

/home/rod/Desktop/test2



/home/rod/Desktop/test2/outcoupling_output/photons_yl.csv

Figure 7.2: a) The OLED simulation; b) The results of an outcoupling simulation.

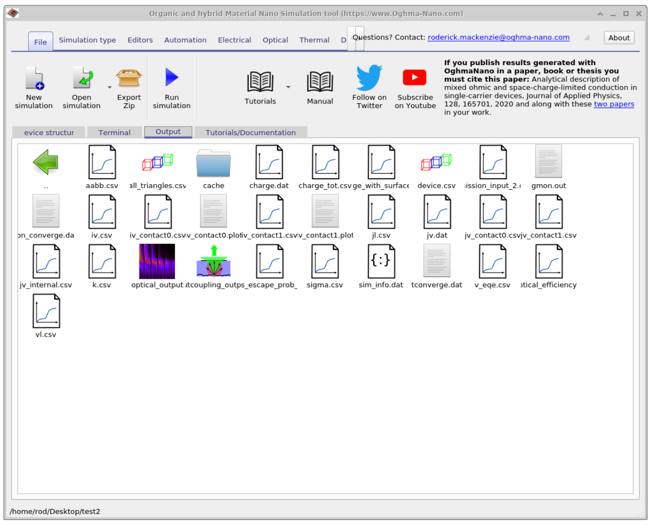


/home/rod/Desktop/test2/outcoupling_output/photons_escape_prob_yl.csv

Figure 7.3: a) Outcoupling calculated using ray the tracing method

7.3 Electrical simulations combined with optical transfer matrix calculations

Make a new simulation based on the *OLED* example as described above. Now run a full simulation by clicking on the *Run simulation* button in the main window. This will run a full optical and electrical OLED simulation one after the other. First an optical simulation will be performed to calculate what the probability of an emitted photon escaping the device will be at each electrical mesh point. Then a full Drift Diffusion simulation will be performed and the optical emission at each mesh point calculated form the recombination. The results will be shown in the *Output* tab of the main window. The key files written to disk are described in table 7.1, examples of these files can be seen in Figures 7.5.



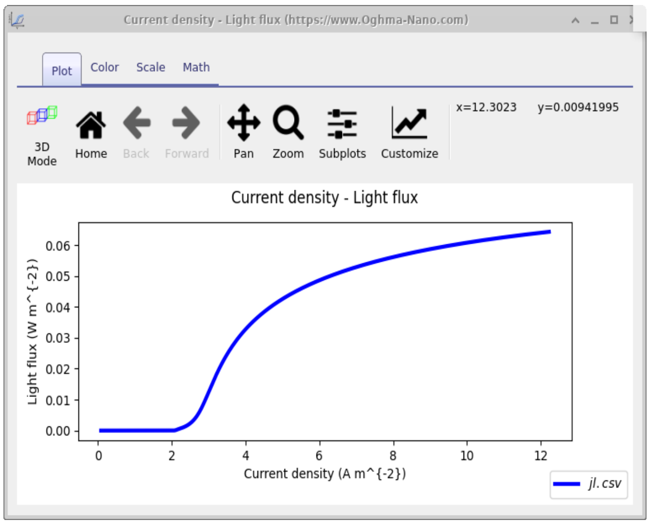
Questions? Contact: roderick.mackenzie@oghma-nano.com

If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

Figure 7.4: a) Out coupling calculated using ray the tracing method; b)

File name	Description
iv.csv	Current v.s. voltage
jv.csv	Current density v.s. voltage
jl.csv	Current density v.s. optical output power density
k.csv	Averaged recombination rate constant v.s. voltage
v_eqe.csv	Voltage v.s. EQE
vl.csv	Voltage v.s. optical output power density
Snapshots	Snapshots of the electrical device parameters for each simulation step

Table 7.1: Key files produced by the OLED simulation.



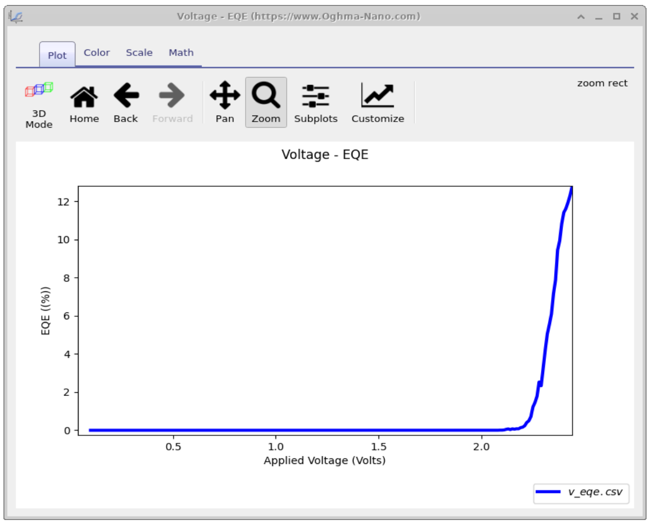
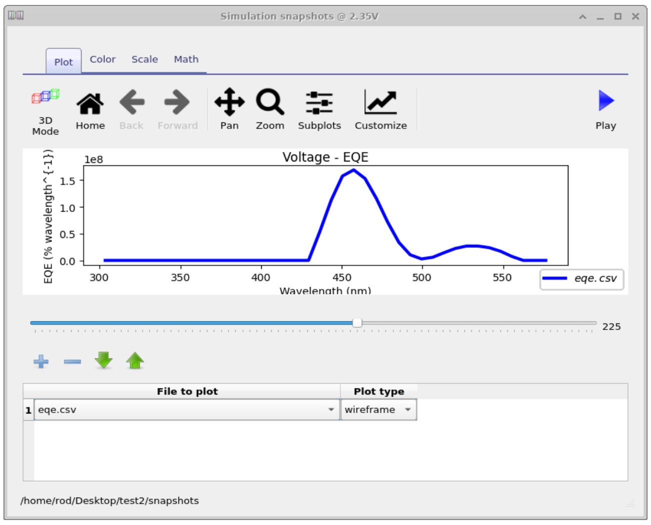


Figure 7.5: Example of the output from the OLED simulations of OghamNano

In the *snapshots* directory you will find detailed electrical output from the model for each simulation step see Figure 7.6. Also in that directory you will find a file entitled *eqe.csv*. This contains the calculated EQE spectrum as a function of voltage. The EQE spectrum will change as the voltage changes because each point in the cavity of the device will have a different probability of outcoupling each wavelength of light to the world. As the carrier distribution in the device changes recombination will happen at different points and thus the overall spectrum as observed from outside the device will change. Use the slider to see how the spectrum changes as a function of voltage.



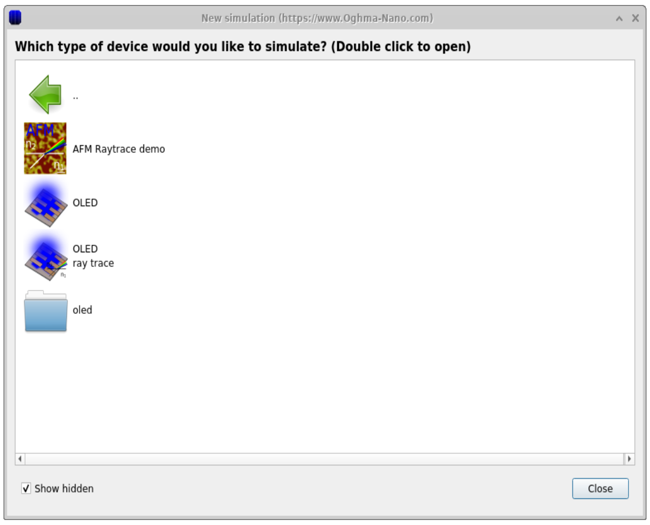
	File to plot	Plot type
1	eqe.csv	wireframe

/home/rod/Desktop/test2/snapshots

Figure 7.6: The EQE spectrum calculated from the device in OghmaNano as a function of wavelength. The graph will change as applied voltage changes. To explore this use the slider in the window scroll over the results for various voltages

7.4 Electrical simulations combined with ray tracing

Ray tracing allows the user to tackle more complex optical problems such as when the device has rough surfaces, it also allows the user to examine rays that don't propagate normally to the interfaces. Two example ray tracing simulations are included *AFM ray trace demo* and *OLED ray trace*, these can be seen in Figure 7.7a and the results of *AFM ray trace demo* can be seen in Figure 7.7b note the rough surface of the OLED in this example.


New simulation (https://www.Oghma-Nano.com)
Which type of device would you like to simulate? (Double click to open)
..
AFM Raytrace demo
OLED
OLED ray trace
oled
Show hidden
Close

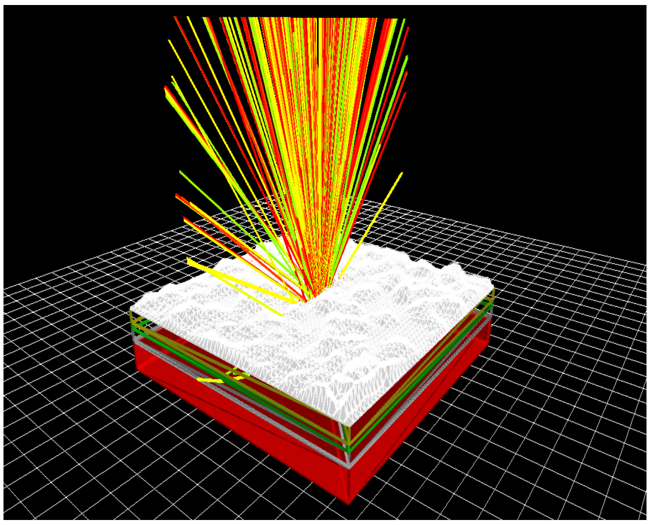


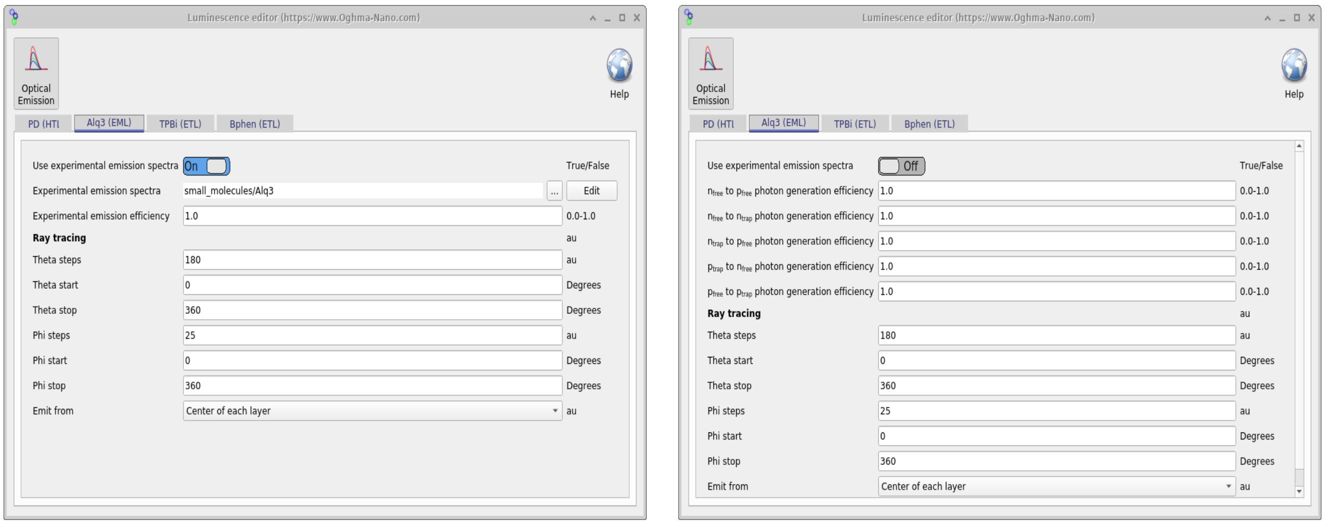
Figure 7.7: a) The new simulation window; b) An example of an OLED ray trace simulation though a rough surface.

7.5 The spectrum of the emitted light

Just as the electrical parameters can be set for each layer in the device so can the emission spectra. This is done in the *emission parameters window* (see Figure 7.8), which can be opened using the *Emission parameters* button in the Device structure tab of the main window (see Figure **??**). The emitted light spectrum can come from one of two sources. The first source is an experimental emission spectra which can be selected by changing the value of *Experimental emission spectra* in 7.8. There are many spectra available in the materials data base but you can also add your own (see the section on databases). By turning the option *Use experimental emission spectra* to *off*, one can use a numerically calculate emission spectra rather than an experimental one. The calculated emission spectra is calculated using Fermi's Golden rule from the carrier population and the DoS. This is only recommended for advanced users most users who are interested in OLED design should use .

If you have selected ray tracing in as the method to be used for outcoupling then for each layer one can select in what direction the rays are emitted. This is under the ray tracing heading visible in Figure 7.8. The values define emission in terms of spherical coordinates theta and phi. One can define the start and stop angles as well as the number of points use for both theta and phi. This enables one to study light escape as a function of angle. It is also worth noting for devices with flat surfaces one only needs to vary phi or theta but not both due to symmetry considerations.

The option *emit from* will tell the ray tracer how to choose the position of the sources for each ray. The option visible in the window is *At the Center of each emission layer*. If this option is selected the model will pick an emission point in the middle of each layer to start the ray tracing from, this is a good option if fast computational times are desired. Another option is *At each electrical mesh point*, this will run the ray tracer at each electrical mesh point to accurately calculate emission probabilities, this can be slower although the decrease in speed can be mitigated against by increasing the number of threads the ray tracer runs on.



Use experimental emission spectra	On	True/False
Experimental emission spectra	small_molecules/Alq3	Edit
Experimental emission efficiency	1.0	0.0-1.0
Ray tracing		au
Theta steps	180	au
Theta start	0	Degrees
Theta stop	360	Degrees
Phi steps	25	au
Phi start	0	Degrees
Phi stop	360	Degrees
Emit from	Center of each layer	au

Use experimental emission spectra	Off	True/False
n_{free} to p_{free} photon generation efficiency	1.0	0.0-1.0
n_{free} to n_{trap} photon generation efficiency	1.0	0.0-1.0
n_{trap} to p_{free} photon generation efficiency	1.0	0.0-1.0
p_{trap} to n_{free} photon generation efficiency	1.0	0.0-1.0
p_{free} to p_{trap} photon generation efficiency	1.0	0.0-1.0
Ray tracing		au
Theta steps	180	au
Theta start	0	Degrees
Theta stop	360	Degrees
Phi steps	25	au
Phi start	0	Degrees
Phi stop	360	Degrees
Emit from	Center of each layer	au

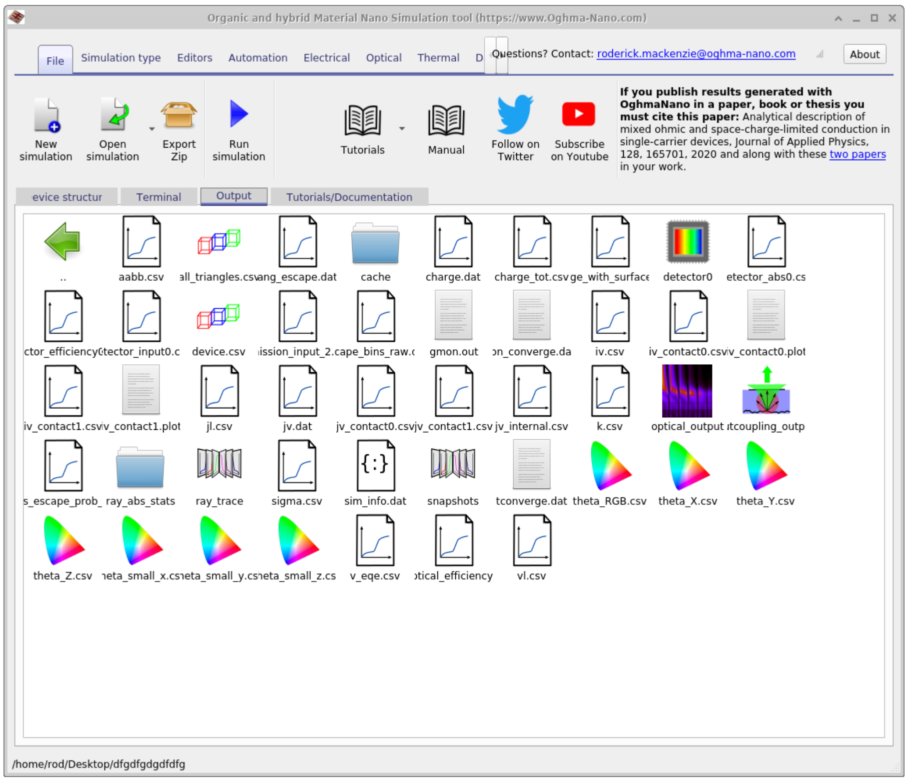
Figure 7.8: a) Using an experimental emission spectra; b) Calculating an emission spectra from the electrical density of states.

7.6 RGB, XYZ, CIE 1931 xyz colour spaces

When designing OLED structures it is often important to understand how color of the emitted light changes as a function of viewing angle. When OghmaNano runs a ray tracing simulation it emits the files in table 7.2 which describe how the color of light changes as a function of viewing angle. These files are also visible in 7.9 as the colored colorspace icons.

File name	Description
theta_RGB.csv	RGB values v.s. viewing angle
theta_x.csv	CIE 1931 x v.s. viewing angle
theta_y.csv	CIE 1931 y v.s. viewing angle
theta_z.csv	CIE 1931 z v.s. viewing angle
theta_X.csv	CIE 1931 X v.s. viewing angle
theta_Y.csv	CIE 1931 Y v.s. viewing angle
theta_Z.csv	CIE 1931 Z v.s. viewing angle

Table 7.2: Files describing the change of color with viewing angle.



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

Figure 7.9: The output of a simulation showing the files described in table 7.2. These files are represented by the color space icons.

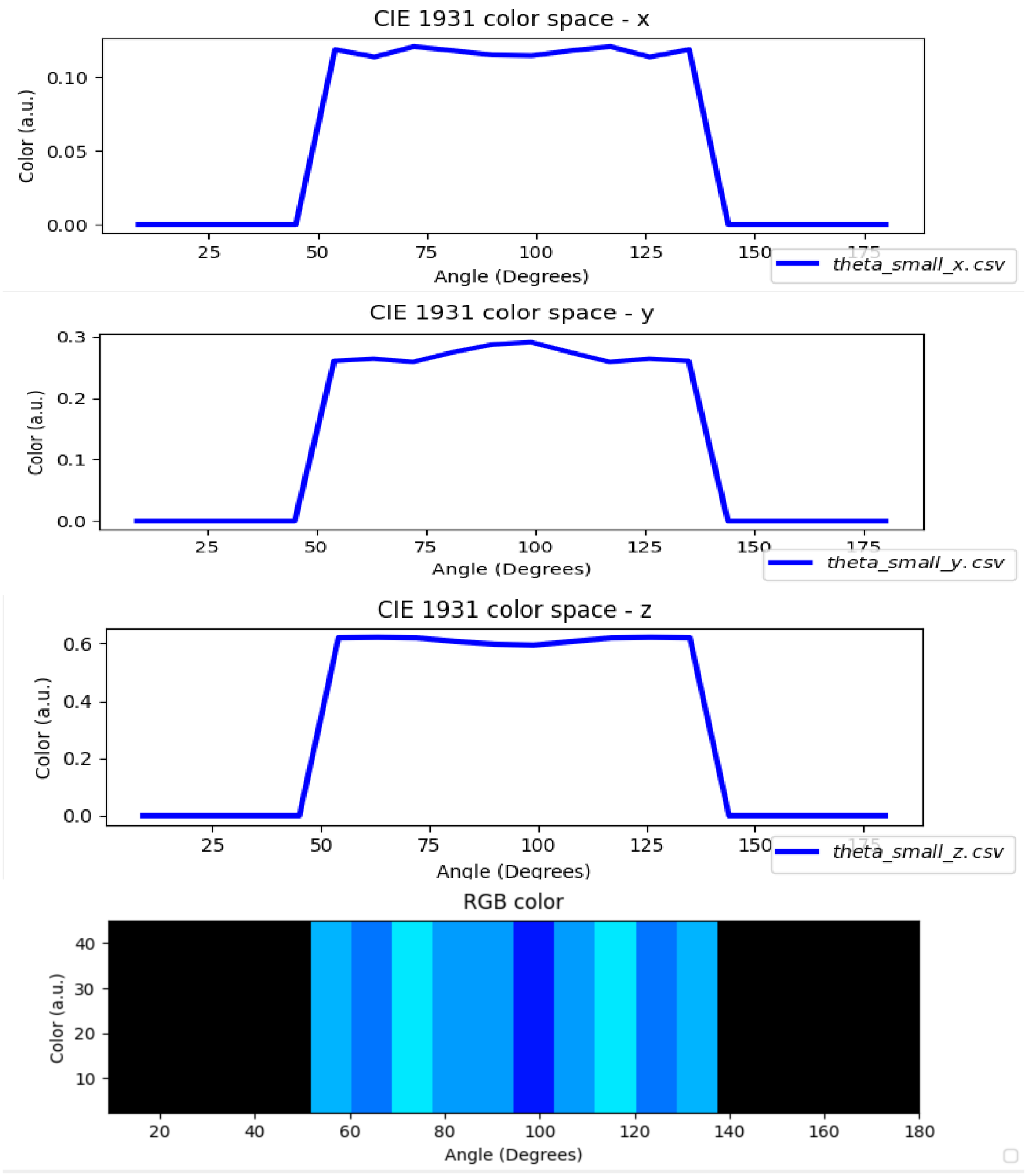


Figure 7.10: Examples of color space output from OghamNano as a function of viewing angle; a) *theta_x.csv*; b) *theta_y.csv*; c) *theta_z.csv*; and $theta_rgb.csv$

Chapter 8

Meshing

8.1 What is meshing?

Meshing is the process of taking a continuous problem space, say a bar along which heat is conducted form a candle to a block of ice and turning this into a discrete computational problem. This usually involves picking a series of points along this space and using these to represent the entire problem (see Figure 8.1). The reason we need to break a region up into points to model it is because computers have finite memory and can not model a continuum very easily. If you want more details why and how this is done search for basic finite difference tutorials on the web.

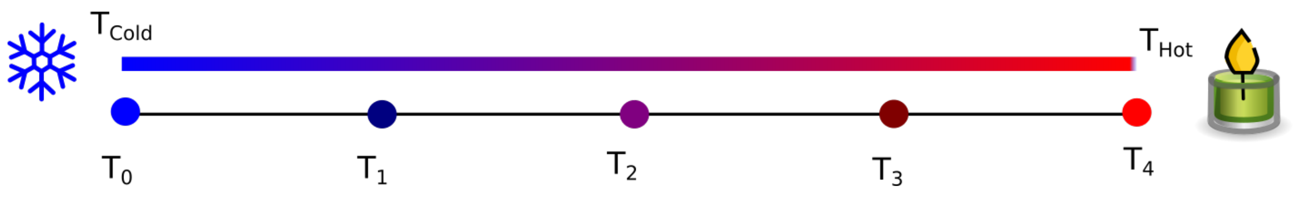


Figure 8.1: An example of a continuous problem broken up (or meshed) into a series of discrete points.

8.2 Different meshes for different problems

In simple terms OghmaNano solves three physical models, the optical model describing light, the thermal model describing thermal effects and the electrical model describing electrical effects. The physical effects from each of these three models often happen on different length scales so need different meshes. For example:

- Electrically interesting effects may only happen in the active layer of a solar cell (100 nm thick), so it is only worth solving the drift diffusion equations in this layer, however light interacts with all layers in the cell (1 μm thick) so the optical problem should be solved over the entire device. This means light should be modelled covering the whole device but electrical effects only in the active layer.

- A Quantum Well laser diode used for cutting steel may generate a lot of heat in it's Quantum Well (30 nm thick) and waveguide layers (1 μm) thick but heat will escape from device through a heat sink which may be a 1 cm thick block of copper. This means electrical effects should be modelled within the device on (1 μm) scale the but thermal effects should be modelled on the cm scale.

Thus different effects need to be simulated on different length scales. Furthermore, you electrical device structure may have some very thin layers such as contact layers or interface layers which are only a few nanometres thick. Optically these layers are far below the wavelength of light so you don't need to worry about them from the optical perspective, but electrically they are very important as they define the current voltage characteristics of the device. Thus you would want to use a very fine electrical mesh over the layer to make sure they were modelled accurately from the electrical stand point but you could use a very wide optical mesh that skips the layers.

In general OghamNano interpolates between the three meshes, so for example if you set up an thermal profile using a temperature mesh, and the temperature values are needed in the electrical problem the values are transferred through interpolation. You don't need to worry about this as a user. The same is true for the optical simulation values of Generation rate etc. are interpolated between from the optical mesh to the electrical mesh as needed.

8.3 The three meshes of OghmaNano

The thermal, optical and electrical ribbons are shown below in Figure 8.2, 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.

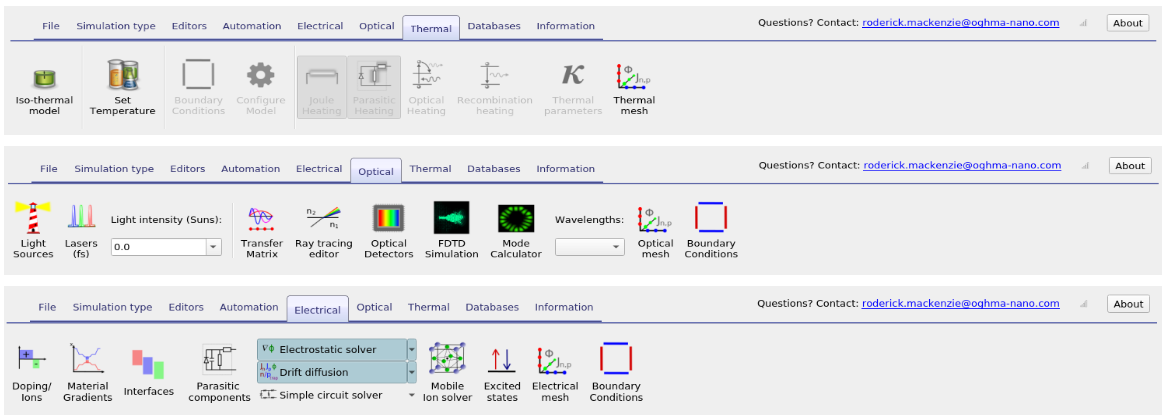
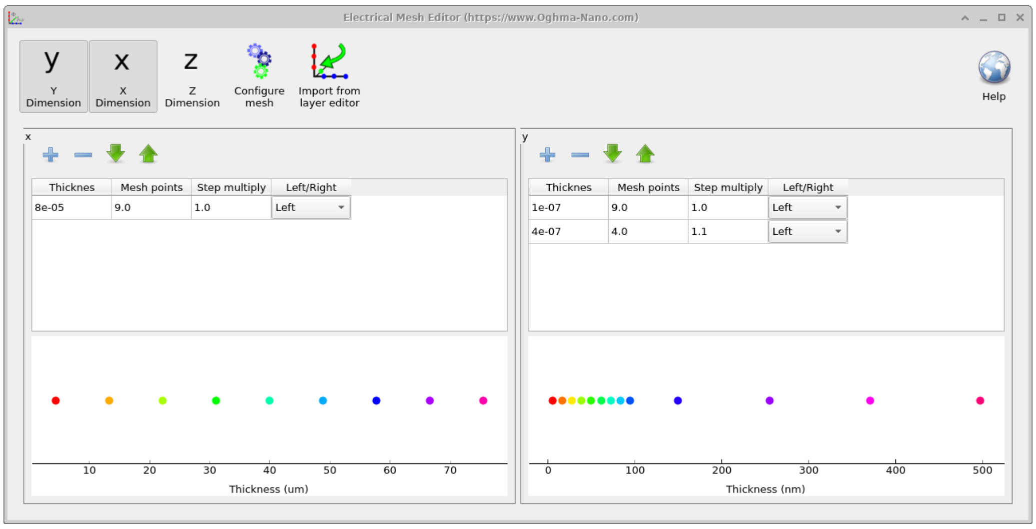


Figure 8.2: The thermal, electrical and optical ribbons.

8.3.1 Electrical mesh

If you click on the electrical mesh button in the electrical ribbon of the main window a window looking like Figure 8.3 will appear. The buttons marked 1D, 2D and 3D at the top of the window in Figure 8.3 can be used to toggle the simulation between 1D, 2D and 3D modes. In this example the mesh is set up for a 2D OFET simulation, so y and x are depressed. 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 tables that looks like a spreadsheet is used to configure the mesh. The columns *thickness* and *mesh points*, determine the thickness of the mesh layer and the number of points on the mesh layer. The column, *step multiply* by how much to grow each step. For the second row of the x mesh the step multiplier is set to 1.1 which means the distance between the points will grow by 10% each mesh point. The buttons marked left/right, defines on which side the mesh layer is generated from. The resulting meshes are plotted in the graphs at the bottom of the window.



Thicknes	Mesh points	Step multiply	Left/Right
8e-05	9.0	1.0	Left

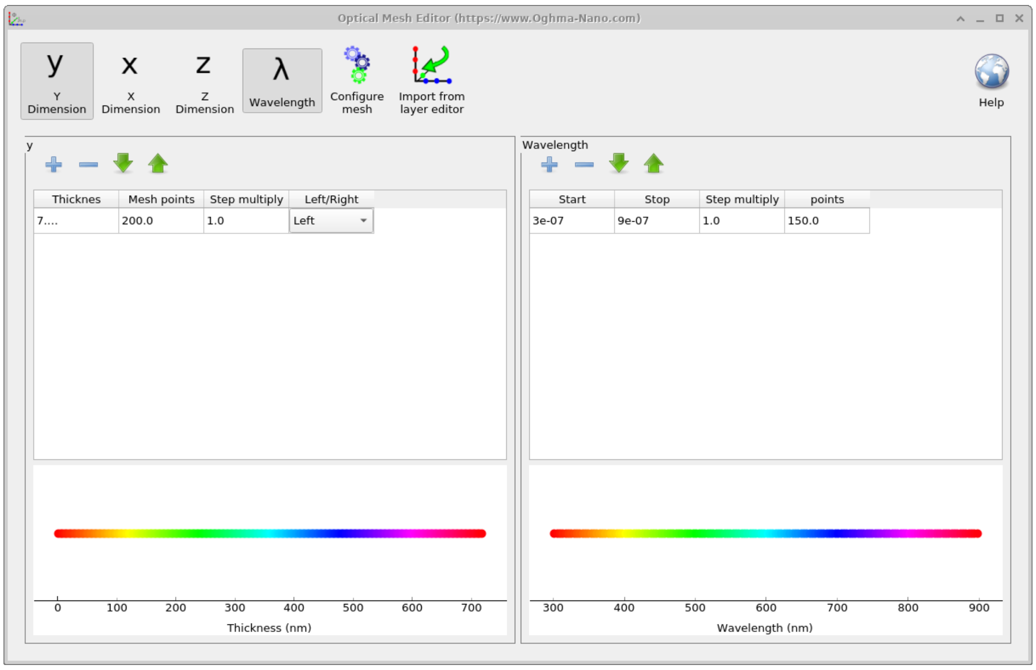
Thicknes	Mesh points	Step multiply	Left/Right
1e-07	9.0	1.0	Left
4e-07	4.0	1.1	Left

Figure 8.3: The electrical mesh editor showing the mesh for a 2D OFET simulation.

The button called *Import from layer editor* clears the y-electrical mesh and imports all the layers from the layer editor giving them four mesh points each. This is useful when setting up complex structures with many layers such as laser diodes.

8.3.2 Optical mesh

The optical mesh window is shown below in Figure 8.4, this is more or less identical to the electrical mesh window except it also has a panel to configure which wavelengths are simulated in a simulation. These wavelengths are used in ray tracing, FTDT simulations and transfer matrix simulations. In any simulation where a wavelength range is defined this mesh will be used.



Thicknes	Mesh points	Step multiply	Left/Right
7....	200.0	1.0	Left

Start	Stop	Step multiply	points
3e-07	9e-07	1.0	150.0

Figure 8.4: The optical mesh editor showing the number of points in position space and the number of wavelength points used for the optical simulations.

8.3.3 Thermal mesh

This will generally be handled automaticity for you. And you will only need to consider the thermal mesh when you turn on self heating in the device. Some parameters such as

8.4 The electrical mesh in detail

A picture of the electrical mesh is given below 8.5 note the mesh does not start at zero but half a mesh point into the device.

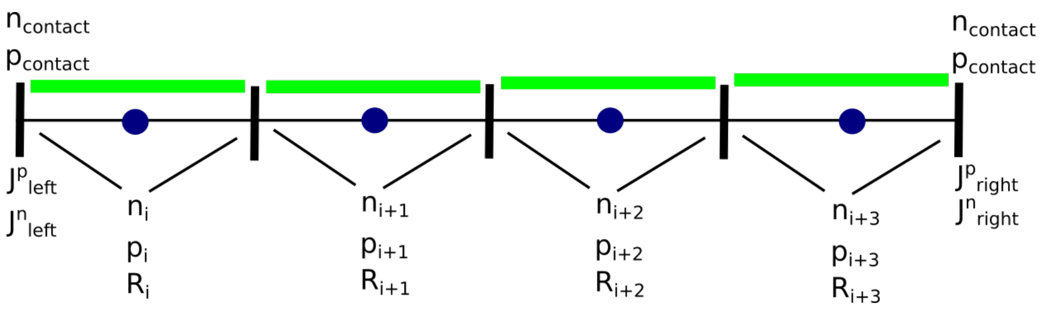


Figure 8.5: A 1D diagram of the mesh

8.5 When do I need to worry about meshing in Ogham-Nano?

8.5.1 Electrical mesh

You will already know that the layer editor is used to split the device up into layers of different materials (See 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 to be setup over the active layer for it to work. The length of the mesh and the length of the active layer must be exactly the same or you will get an error. OghmaNano will try to do this for you in most cases, so generally speaking for simple problems you will not have to think about meshing. However if one starts adding multiple active layers you may have to define your own mesh. Other times when you will need to worry about meshing is when you are interested in cutting down the number of mesh points used to speed your simulation up or when you want to make your simulation more accurate by increasing the number of mesh points.

8.5.2 Optical mesh

You will need to play with the optical mesh if you want to change the wavelength range over which light is simulated in the device. You will also want to change the number of points in position space in this mesh if you want more accurate (more mesh points) or faster (more mesh points) optical simulations.

8.5.3 Thermal mesh

If you want to do simulations with trap states with self heating turned on you will need to define a thermal mesh, otherwise this will be taken care of for you.

8.6 Meshing tips

8.6.1 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 micros 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 an 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 dimentions
$Solarcells$	1D
$Opticalfilter$	1D
$OFET$	2D

Table 8.1: How many dimensions should I use to simulate my device.

8.6.2 Speed v.s. accuracy

When setting up your device in OghmaNano you should try to:

- Minimize the number of mesh points to improve computation speed but not go so low that accuracy is sacrificed.
- Realise that more mesh points does not necessarily mean a more accurate simulation. Especially in the electrical model the equations are discretized as not to lineally interpolate between mesh points, instead the actual differential equations are solved as boundary value problems between the mesh points. This means you can sometimes get away with surprisingly few mesh points (very fast simulations) and still get quite nice results.
- Don't be afraid to really reduce the number of mesh points when testing out fitting or trying to understand device performance. This will give you a very fast simulation. You can always go back later and add more mesh points.

Chapter 9

Theory of drift diffusion modelling

9.1 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.

9.2 Electrostatic potential

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

$$E_{LUMO} = -\chi - q\phi \tag{9.1}$$

$$E_{HOMO} = -\chi - E_g - q\phi \tag{9.2}$$

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

$$\nabla \cdot \epsilon_0 \epsilon_r \nabla = q(n_f + n_t - p_f - p_t - N_{ad} + -N_{ion} + a), \tag{9.3}$$

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.

9.3 Free charge carrier statistics

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

$$n_l = N_c exp\left(\frac{F_n - E_c}{kT}\right) \tag{9.4}$$

$$p_l = N_v exp\left(\frac{E_v - F_p}{kT}\right) \tag{9.5}$$

or full Fermi-dirac statistics i.e.

$$n_{free}(E_f, T) = \int_{E_{min}}^{\infty} \rho(E) f(E, E_f, T) dE \tag{9.6}$$

$$p_{free}(E_f, T) = \int_{E_{min}}^{\infty} \rho(E) f(E, E_f, T) dE \tag{9.7}$$

where

$$f(E) = \frac{1}{1 + e^{E - E_f/kT}} \tag{9.8}$$

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

$$\rho(E)_{3D} = \frac{\sqrt{E}}{4\pi^2}\left(\frac{2m^*}{\hbar^2}\right)^{3/2} \tag{9.9}$$

The average energy of the carriers is defined as

$$\bar{W}(E_f, T) = \frac{\int_{E_{min}}^{\infty} E\rho(E) f(E, E_f, T) dE}{\int_{E_{min}}^{\infty} \rho(E) f(E, E_f, T) dE} \tag{9.10}$$

9.4 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 9.11, 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 9.11. 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 9.4.2. In fact, it is turned on by default so when using OghmaNano you have to go out of your way to turn on equation 9.11.

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 1-2μs 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 9.11 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

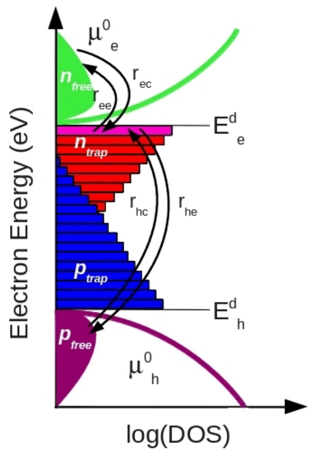


Figure 9.1: Trap filling in both energy and position space as the solar cell is taken from a negative bias Carrier trapping, de-trapping, and recombination

ordered device turn ON the non-equilibrium solver.

9.4.1 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:

$$R^{SRH} = \frac{np - n_0 * p_0}{\tau_p(n+n_1) + \tau_n(p+p_1)} \tag{9.11}$$

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.

9.4.2 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 9.1, 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 9.1. The rate rec 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 9.12 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 9.1.

$$\frac{\delta n_t}{\partial t} = r_{ec} - r_{ee} - r_{hc} + r_{he} \tag{9.12}$$

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 9.1: 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:

$$e_n = v_{th}\sigma_n N_c exp\left(\frac{E_t - E_c}{kT}\right) \tag{9.13}$$

and

$$e_p = v_{th}\sigma_p N_v exp\left(\frac{E_v - E_t}{kT}\right) \tag{9.14}$$

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

$$\rho^{e/h}(E) = N^{e/h} exp(E/E_u^{e/h}) \tag{9.15}$$

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:

$$N_t(E) = \frac{\int_{E-\Delta E/2}^{E+\Delta E/2} \rho^e E dE}{\Delta E} \tag{9.16}$$

The occupation function is given by the equation,

$$f(E_t, F_t) = \frac{1}{e^{\frac{E_t - F_t}{kT}} + 1} \tag{9.17}$$

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

9.5 Free-to-free carrier recombination

A free-carrier-to-free-carrier recombination (bi-molecular) pathway is also included. Free-to-free recombination is described using equation 9.18

$$R_{free} = k_r(n_f p_f - n_0 p_0) \tag{9.18}$$

in some situations where one is trying to fit rate equations to the model it can be useful to have equation 9.18 written in another form,

$$R_{free} = k_r(n_f p_f - n_0 p_0)^{\frac{\lambda+1}{2}} \tag{9.19}$$

this can be turned on using the option called *Enable λ power in free to free recombination.* in the configure window of the Electrical parameter editor.

Free to free recombination is equivalent to Langevin recombination. 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. See section 9.12 for a general discussion on the need for trap states and why generally Langevin recombination should not be used in organic solar device models.

9.6 Auger recombination

Auger recombination is as

$$R^{AU} = (C_n^{AU} n + C_p^{AU} p)(np - n_0 p_0) \tag{9.20}$$

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.

9.7 Charge carrier transport

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

$$\boldsymbol{J_n} = q\mu_e n_f \nabla E_c + qD_n \nabla n_f, \tag{9.21}$$

and holes,

$$\boldsymbol{J_p} = q\mu_h p_f \nabla E_v - qD_p \nabla p_f. \tag{9.22}$$

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

$$\nabla \boldsymbol{J_n} = q(R - G + \frac{\partial n}{\partial t}), \tag{9.23}$$

and holes

$$\nabla \boldsymbol{J_p} = -q(R - G + \frac{\partial p}{\partial t}). \tag{9.24}$$

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

9.8 Perovskite mobile ion solver

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

$$\boldsymbol{J_a} = q\mu_a a_f \nabla E_v - qD_a \nabla a_f. \tag{9.25}$$

$$\nabla \boldsymbol{J_a} = -q\frac{\partial a}{\partial t}. \tag{9.26}$$

9.9 Semiconductor interfaces

9.9.1 Tunnelling through heterojunctions

Tunnelling of holes through hetrojunction interfaces are is give by

$$\boldsymbol{J_p} = qT_h((p_1 - p_1^{eq}) - (p_0 - p_0^{eq})), \tag{9.27}$$

and for electrons

$$\boldsymbol{J_n} = -qT_e((n_1 - n_1^{eq}) - (n_0 - n_0^{eq})). \tag{9.28}$$

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

9.9.2 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.

9.10 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

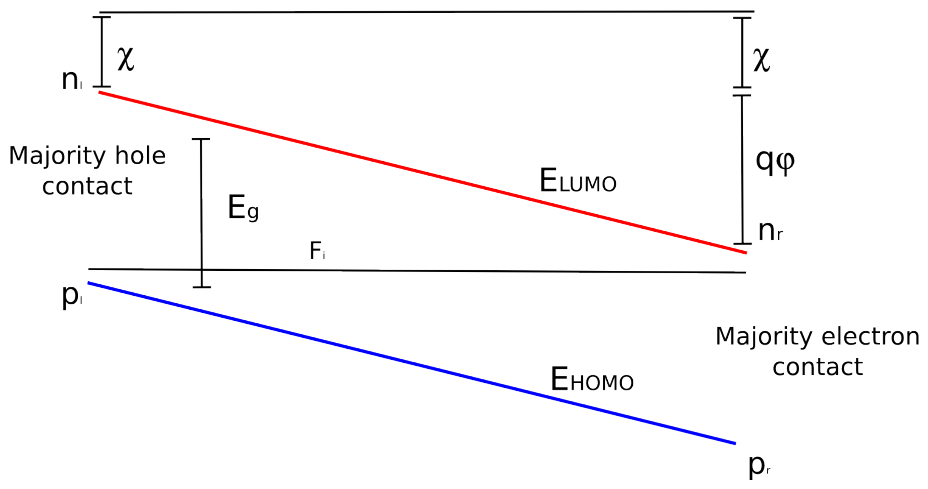


Figure 9.2: Band structure of device in equilibrium.

The left hand side of the device is given a reference potential of 0 V. See figure 9.2. We can then write the energy of the LUMO and HOMO on the left hand side of the device as:

$$E_{LUMO} = -\chi \tag{9.29}$$

$$E_{HOMO} = -\chi - E_g \tag{9.30}$$

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

$$p_l = N_v exp\left(\frac{E_{HOMO} - F_p}{kT}\right) \tag{9.31}$$

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

$$n_l = N_c exp\left(\frac{F_n - E_{LUMO}}{kT}\right) \tag{9.32}$$

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 *phi* to take account of the built in potential.

$$E_{LUMO} = -\chi - q\phi \tag{9.33}$$

$$E_{HOMO} = -\chi - E_g - q\phi \tag{9.34}$$

we can now calculate the potential using

$$n_r = N_c exp\left(\frac{F_n - E_{LUMO}}{kT}\right) \tag{9.35}$$

equation 9.33.

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

$$p_r = N_v exp\left(\frac{E_v - F_{HOMO}}{kT}\right) \tag{9.36}$$

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.

9.10.1 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.:

$$\mu_e(n) = \frac{\mu_e^0 n_{free}}{n_{free} + n_{trap}} \tag{9.37}$$

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.

9.11 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 $1x10^{-5}$
- 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 $1x10^{-5}$
- 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:

9.11.1 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 $1.000000000000000e19$ and not $1.0000000000000000001e19$, 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 wills 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.

9.12 The need for trap states in device organic models

Related YouTube videos:

Please stop simulating disordered semiconductors without trap states.

This section explains why trap states need to be considered when simulating disordered materials such as polymer/small molecule devices. It also touches on why using full SRH recombination/trapping model is so important to get physically meaningful results from a device model.

9.12.1 The physical and energetic structure of disordered materials.

Traditional inorganic semiconductor such as crystalline Si or GaAs are highly ordered and are almost completely pure it is not uncommon to get a material that is nine nines pure or, 99.9999999% pure. Organic semiconductors on the other hand are very really quite dirty with purities often around 99.9% which is six orders of magnitude more dirty than their inorganic counterparts, thus they have around a million times more impurities than their inorganic counterparts. Added to this inorganic semiconductors are highly ordered with a regular crystalline structure one can think of them as marbles packed on a solitaire board (see Figure 9.3), while organic semiconductors are a floppy mess of molecules which one can think of more as spaghetti bolognese with the spaghetti representing the polymers and the bolognese representing small molecules (see Figure 9.4).

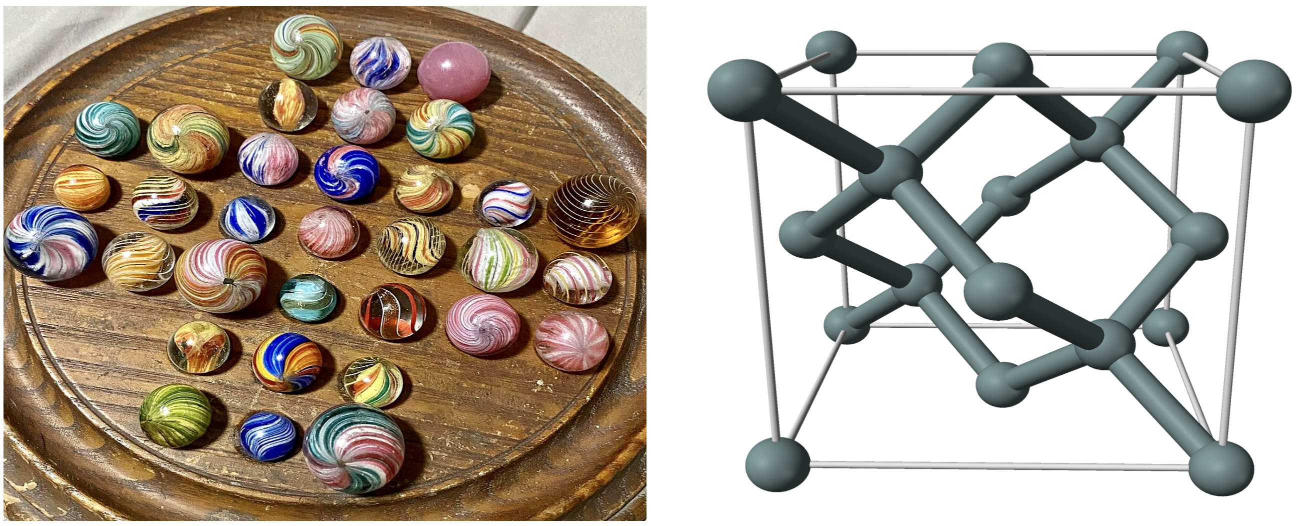


Figure 9.3: Left) Marbles in an ordered arrangement on a solitaire board [12]; Right) Silicon atoms ordered within a material[13] Both systems are highly ordered.

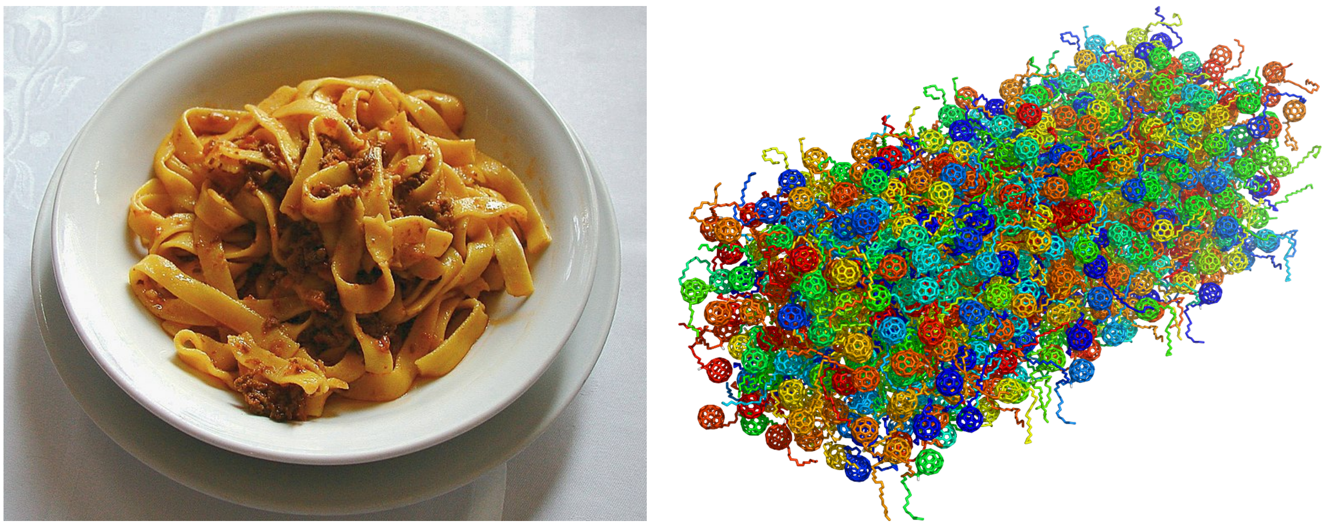


Figure 9.4: Left) A plate of spaghetti [14]; Right) A polymer packing like spaghetti. Both systems are highly disordered.

So on one hand we have an organic material that is messy and highly disordered, and on the other hand we have a material such as silicon that is highly pure and very ordered. This physical differences results in a very different energetic landscape for the two materials. In the ordered material semiconductor electrons/holes can travel freely in the conduction and valance bands. If an electric field is applied they only experience a small resistive force. Such a band structure is shown in Figure 9.5a. In the organic material the picture is very different, due to the disorder and impurities the band structure is full trap states. There are so many trap states that the carriers no longer move freely but hop between the trap states after being thermally excited, such a band structure is shown in shown in Figure 9.5b. Thus there are two very different charge transport mechanisms in these two materials.

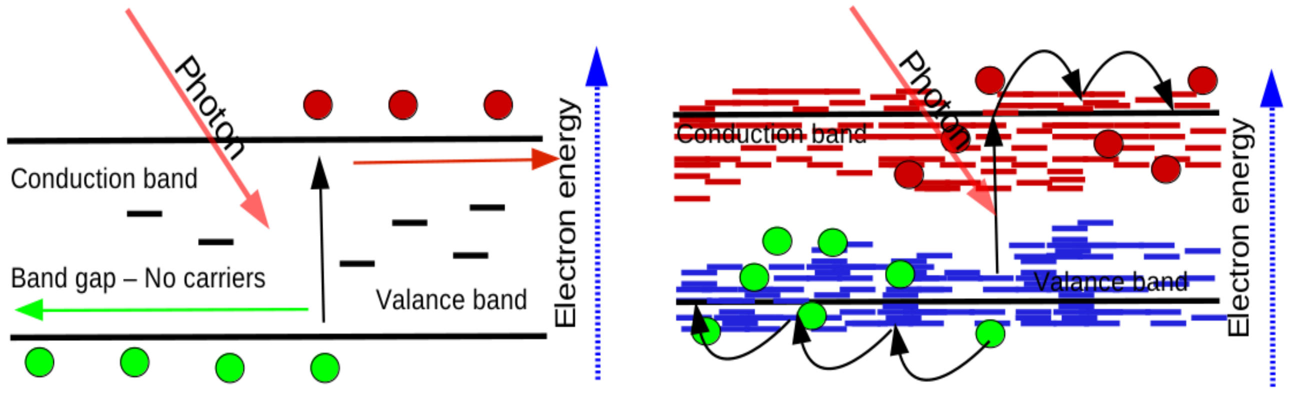


Figure 9.5: a) The band structure of an ordered semiconductor such as GaAs; b) The band structure of an disordered material such as PM6:Y6 or P3HT:PCBM.

9.12.2 Trap states and charge density

[This section needs improving/editing but the sketch of what it should say is there:] Figure 9.6 sketches out the distribution of states for Figure 9.5. On the left of the image is a ordered semiconductor with a parabolic band structure. The Fermi distribution of electrons is coloured in purple. The right hand side image shows a disordered semiconductor with an exponential density of trap states going into the band gap (sometimes a Gaussian DoS is used). It can be seen that the DoS and the distribution/energetic position of charge carriers are is very different between the two types of semiconductor.

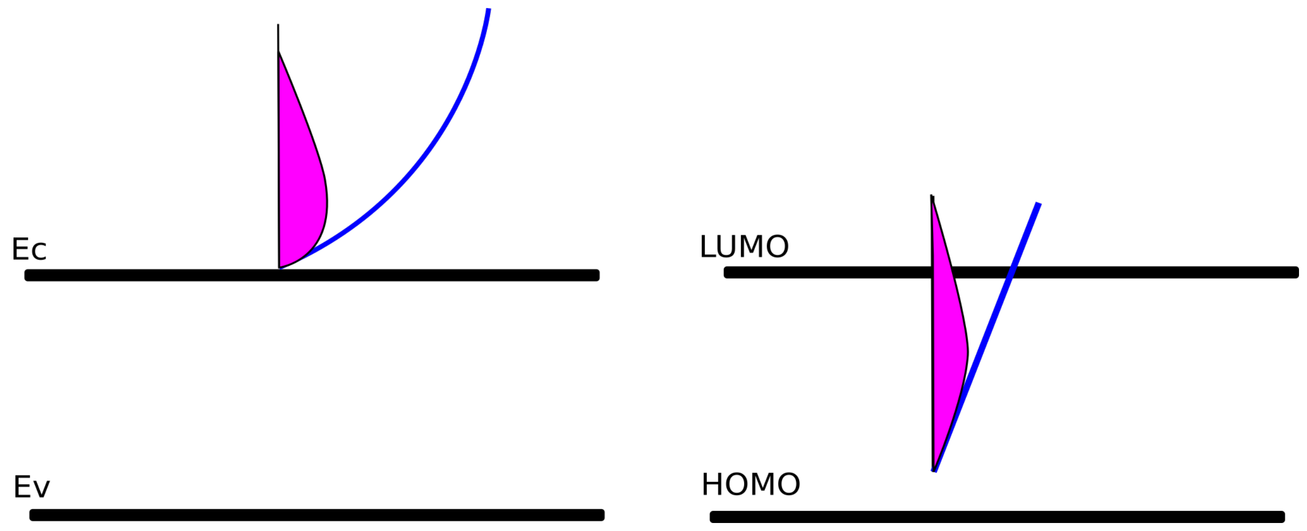


Figure 9.6: a) The band structure of an ordered semiconductor such as GaAs; b) The band structure of an disordered material such as PM6:Y6 or P3HT:PCBM.

The total charge density at any place in the device can be described by an integration of the Fermi-Dirac function, and the DoS ρ.

$$n(E_f,T) = \int_{E_{min}}^{\infty} \rho(E) f(E,E_f) dE \tag{9.38}$$

Where E_f is the Fermi level. Clearly ρ will be very different for an ordered and a disordered semiconductor. Thus the dependence of $n(E_f,T)$ on E_f and thus applied voltage will very depending on what ρ is chosen for the device. In practical terms this means that a disordered device will have a lot more traps closer to the Fermi level and thus for any given voltage it will contain one or two orders of magnitude more charge than an ordered device, this can be observed in Charge Extraction experiments. So if one ignores trap states when modelling a disordered device then the function $n(Voltage)$ will be wrong.

If $n(Voltage)$ is not correct then the recombination rate will be wrong for any given voltage:

$$R = k_r n(Voltage) p(Voltage) \tag{9.39}$$

Furthermore if $n_{free}(E_f,T)$ is wrong the mobility will also have an incorrect dependence on voltage:

$$\mu_e(n) = \frac{\mu_e^0 n_{free}}{n_{free} + n_{trap}} \tag{9.40}$$

So if your DoS is wrong (i.e. no traps). Then you have no chance of reproducing a JV curve correctly. Summary: OghmaNano was written specifically to simulate disordered devices where trap states are play a large role in transport and recombination. Examples of such materials are PM6:Y6 and P3HT:PCBM. OghmaNano includes traps correctly, make sure what ever model you are using also includes traps or it will be wrong.

9.12.3 Why you should not use Langevin recombination in device models

Langevin recombination is defined as,

$$R_{free} = qk_r \frac{(\mu_e + \mu_h)}{2\epsilon_0\epsilon_r} np \tag{9.41}$$

where R_{free} is the recombination rate, k_r is the Langevin reduction factor and all other symbols have their usual meaning. In general Langevin recombination is a bad way to describe recombination in OPV devices. There were some older papers from the early 2010s using this mechanisum but the models could not self consistently describe dark and light JV curves. 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. It was often found that the Langevin equation could not reproduce the experimental results and predicted recombination rates far higher than were experimental observed. To account for this a Langevin reduction factor k_r was often introduced into the equation, and a lot of effort went into measuring k_r. This need for a reduction factor pointed at some deeper issues with the equation.

If we look at the equation for Langevin recombination we can immediately see some issues with it. The first thing we notice is that R_{free} can only ever change as the square of the charge density (n p), but we know from experiment R_{free} can is often a higher order than 2 e.g. $(np)^{1.5}$. Furthermore we can see two mobility terms, however we know from the discussion from above that mobility is a function of carrier density. So the fact that it has the wrong dependence on carrier density and needs a reduction factor points at the mechanism on which it is based being incorrect, and using it will always be like trying to get a square peg in a round hole.

9.12.4 How one can make Langevin recombination work in device models

So they key problems with Langevin recombination are a wrong dependence on carrier density and the need for a reduction factor. It is possible to make Langevin recombination 'work' by making the charge carrier mobilities a function of carrier density as was done in [15]:

$$R_{free} = qk_r \frac{(\alpha\mu_e(n) + \beta\mu_h(n))n_{tot}p_{tot}}{2\epsilon_0\epsilon_r} \tag{9.42}$$

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

$$\mu_e(n) = \frac{\mu_e^0 n_{free}}{n_{free} + n_{trap}} \tag{9.43}$$

and

$$\mu_h(n) = \frac{\mu_h^0 p_{free}}{p_{free} + p_{trap}} \tag{9.44}$$

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

$$R(n,p) = qk_r \frac{(\alpha\mu_e^0 n_{free}p_{trap} + \beta\mu_h p_{free}n_{trap})}{2\epsilon_0\epsilon_r} \tag{9.45}$$

Thus by making the mobility carrier density dependent we arrive at an expression for Langeving 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 we have 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 is 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 a better description of recombination/trapping.

9.13 Thermal models

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 9.13.1, 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 9.13.2 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 9.21 and 9.25 are only valid in isothermal conditions. The full transport equations as derived from the BTE [16] are given by

$$\mathbf{J}_n = \mu_e n \nabla E_c + \frac{2}{3}\mu_e n \nabla \bar{W} + \frac{2}{3}\bar{W}\mu_e \nabla n - \mu_e n \bar{W}\frac{\nabla m_e^*}{m_e^*} \tag{9.46}$$

$$\mathbf{J}_p = \mu_h p \nabla E_v - \frac{2}{3}\mu_h p \nabla \bar{W} - \frac{2}{3}\bar{W}\mu_h \nabla p + \mu_p p \bar{W}\frac{\nabla m_h^*}{m_h^*} \tag{9.47}$$

where $\bar{W}$ is the average kinetic energy of the free carriers as given by 9.10. If the average energy is assumed to be 3/2kT, 9.46 and 9.47, 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 9.7. 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”.

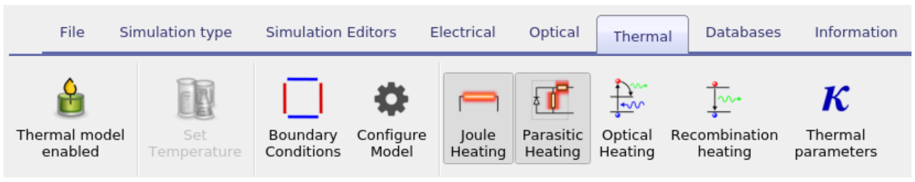


Figure 9.7: Thermal

9.13.1 Lattice thermal model

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

$$0 = \nabla \kappa_l \nabla T_L + H_j + H_r + H_{optical} + H_{shunt} \tag{9.48}$$

where joule heating (H_j) is give by

$$H_j = J_n \frac{\nabla E_c}{q} + J_h \frac{\nabla E_h}{q}, \tag{9.49}$$

recombination heating (H_r) is given by,

$$H_r = R(E_c - E_v) \tag{9.50}$$

optical absorption heating is given by,

$$H_{optical} \tag{9.51}$$

and heating due to the shunt resistance is given by

$$H_{shunt} = \frac{J_{shunt} V_{applied}}{d}. \tag{9.52}$$

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

9.13.2 Energy balance - hydrodynamic transport model

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

$$H = \frac{3k_b}{2}\left(n(\frac{T_n - T_l}{\tau_e}) + p(\frac{T_p - T_l}{\tau_h})\right) + R(E_c - E_v) \tag{9.53}$$

and the energy transport equation for electrons

$$S_n = -\kappa_n \frac{dT_n}{dx} - \frac{5}{2}\frac{k_b T_n}{q} J_n \tag{9.54}$$

and holes,

$$S_p = -\kappa_p \frac{dT_p}{dx} + \frac{5}{2}\frac{k_b T_p}{q} J_p \tag{9.55}$$

will be solved.
The energy balance equations will also be solved for electrons,

$$\frac{dS_n}{dx} = \frac{1}{q}\frac{dE_c}{dx} J_n - \frac{3k_b}{2}\left(RT_n + n(\frac{T_n - T_l}{\tau_e})\right) \tag{9.56}$$

and for holes

$$\frac{dS_p}{dx} = \frac{1}{q}\frac{dE_v}{dx} J_p - \frac{3k_b}{2}\left(RT_p + n(\frac{T_p - T_l}{\tau_e})\right) \tag{9.57}$$

The thermal conductivity of the electron gas is given by

$$\kappa_n = \left(\frac{5}{2} + c_n\right)\frac{{k_b}^2}{q} T_n \mu_n n \tag{9.58}$$

and for holes as,

$$\kappa_p = \left(\frac{5}{2} + c_p\right)\frac{{k_b}^2}{q} T_p \mu_p p \tag{9.59}$$

Chapter 10

Optical models

10.1 Light sources

In OghmaNano light comes from light sources. Light sources defined in the light source editor this can be found in the Optical ribbon of the main window see figure 10.1.

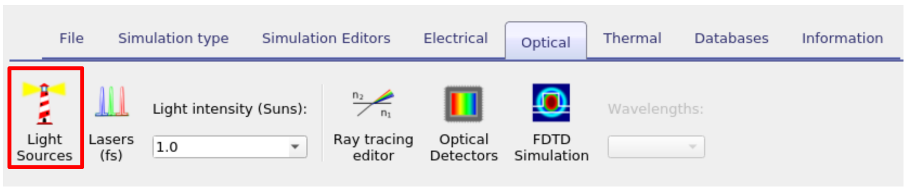


Figure 10.1: Opening the light source editor.

The light source editor is shown below in Figure 10.2. Each light source consists of of illumination spectra and optical filters. In this way you can define a light source to emit AM1.5G but then filter out various components of the spectrum. This enables one to simulate for example a device under AM1.5G spectra but behind a thick glass contact that takes sunlight below 300 nm. Light sources can be combined in the *Light source* tab, so for example one could combine AM1.5G and light given off by a fluorescent tube. Each spectrum is multiplied by a *multiplier* defined in the *multiplier* column this defines the relative intensity of the light sources.

In the filters tab you can define the optical filters. They can be enabled or disabled using the Enable switch, you can select the material which will act as a filter, and decide how much the filter attenuates in dB.

The configure tab of each filter can be used to select where the light comes from, options are:

1. Top: Light will come form the top of the device. (y0) This is usually used with the transfer matrix solver. See the bottom left of figure 10.3.

2. Bottom: Light will come from the bottom of the device. (y1) This is usually used with the transfer matrix solver. See the bottom right of figure 10.3.

3. xyz: The light can come from an xyz position in the simulation space this used for FDTD simulations or ray tracing simulations. This can be seen in to bottom right of figure 10.3.

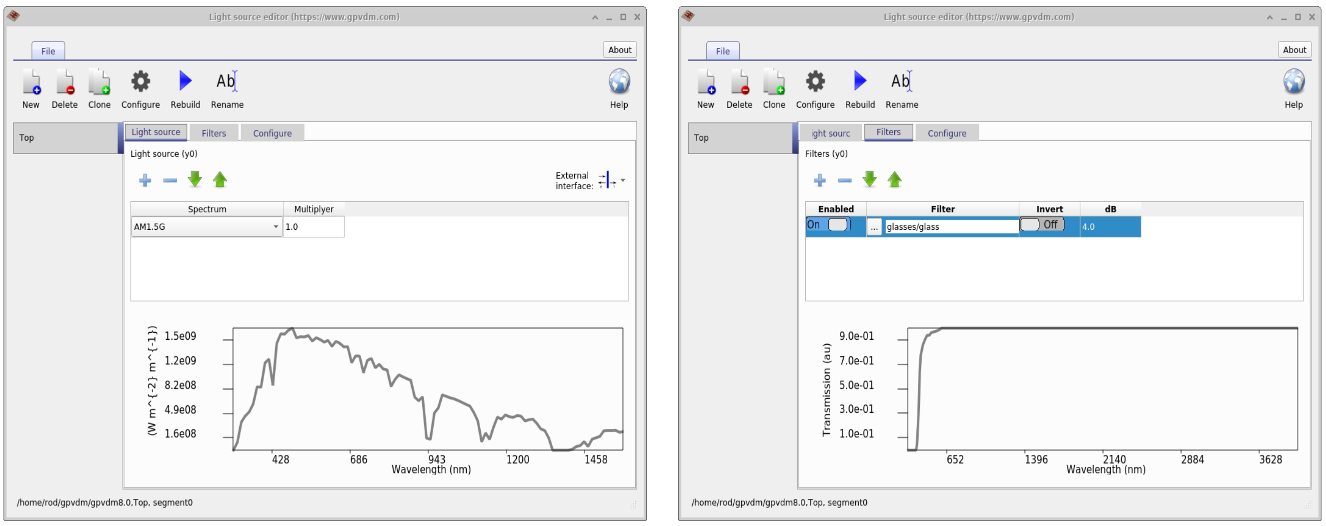
Stop and start wavelengths can also be set in the configure tab.

10.1.1 Local ground view factor

The local ground view factor which is given as [17]

$$F_{ground} = sin^2\left(\frac{\theta_t}{2}\right) = \frac{1 - cos(\theta_t)}{2} \tag{10.1}$$

can be set in the configure tab.



Light source (y0)

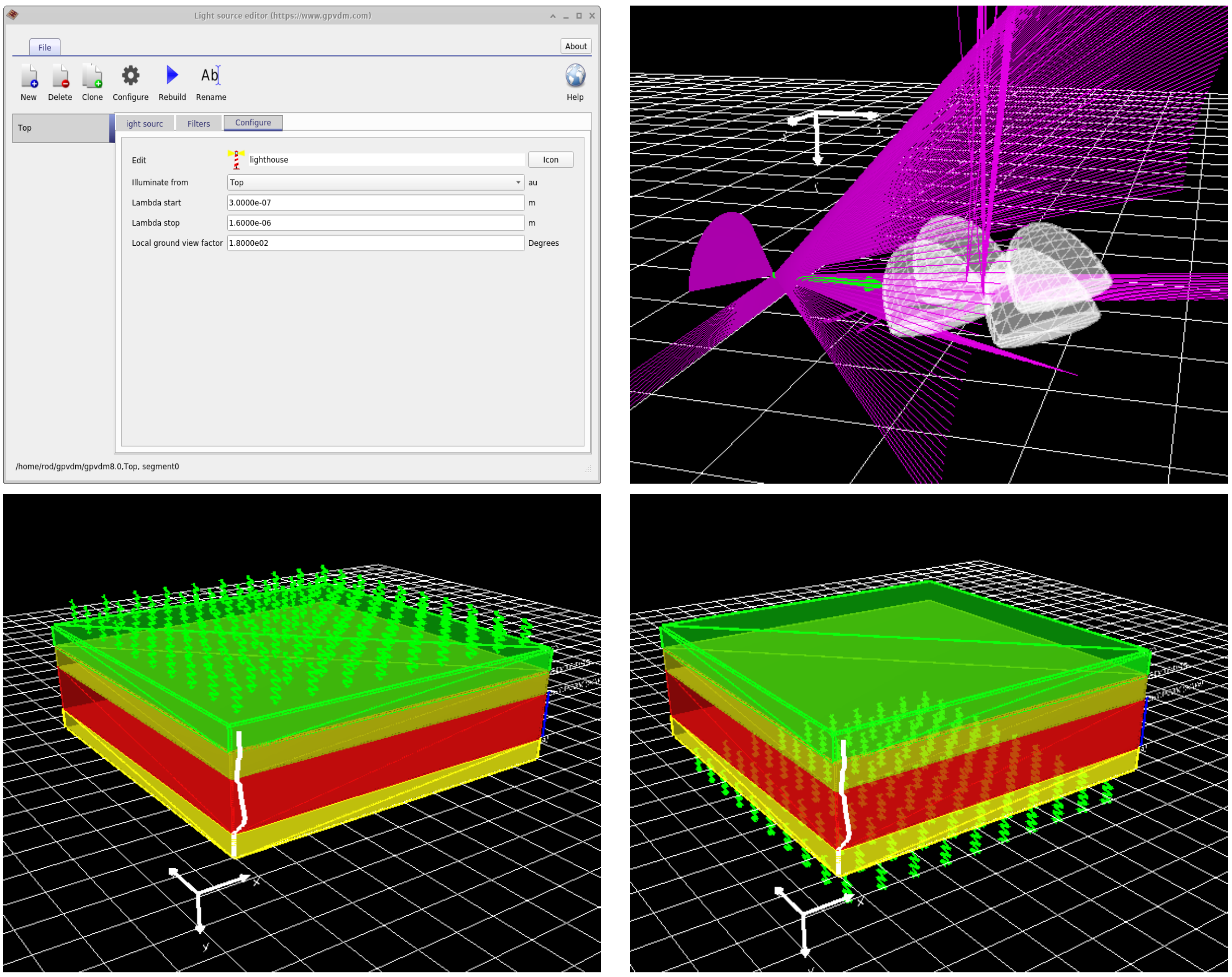
Spectrum	Multiplyer
AM1.5G	1.0

Filters (y0)

Enabled	Filter	Invert	dB
On	glasses/glass	Off	4.0

/home/rod/gpvdm/gpvdm8.0,Top, segment0

Figure 10.2: Left: Building an optical spectrum; Right modifying the light sources with optical filters.



Edit lighthouse

Illuminate from Top au

Lambda start 3.0000e-07 m

Lambda stop 1.6000e-06 m

Local ground view factor 1.8000e02 Degrees

/home/rod/gpvdm/gpvdm8.0,Top, segment0

Figure 10.3: Top left: The configure panel of the light source editor; Top right: Illuminate from set to ”xyz”; bottom left: Illuminate from set to ”top”; bottom right: Illuminate from set to ”bottom”.

10.2 Transfer matrix model

In general the transfer matrix method is good for simulating sandwich type structures where light is incident normal to the surface of a device. Examples of such devices are solar cells, optical filters or sensors. This method is good for understanding optical absorption, reflection and transition in these structures. There are other more complex methods which can accomplish the same task such as FDTD, however general speaking the transfer matrix model will be orders of magnitude faster than these methods.

10.2.1 The user interface

The transfer matrix simulation tool can be reached from the *Optical ribbon* in the main window by selecting *Transfer matrix*. If you click *Run optical simulation* (see 10.4) the distribution of light within the structure will be calculated as a function of position and wavelength. You can see from the top of the figure that there are various simulation modes. The full transfer matrix method is selected by selecting *Transfer matrix*, this will do full optical simulation. The other buttons represent other simplified approximations to the transfer matrix method that allow the user to explore more simple charge carrier generation profiles. From the left the push buttons in Figure 10.4 represent the following optical models:

- *Transfer matrix:* This is a full transfer matrix simulation that takes into account multiple reflections from the interfaces and optical loss within the structure. This is in effect solving the wave equation in 1D and is an accurate (and recommended) optical model to use. See section 10.2.4. If you are unsure which model to pick, pick this one.

- *Exponential profile*: This is a very simple optical model that assumes light decays exponentially according to the relation
$$I = I_0 e^{-\alpha x} \tag{10.2}$$
between layers and and assumes light is transmitted between layers according to the formula:
$$T = 1.0 - \frac{n_1 - n_0}{n_1 + n_0} \tag{10.3}$$
No reflection is accounted for.

- *Flat profile*: The flat profile assumes light is constant within layers and only decreases at material interfaces according to equation 10.3. One might want to use this model when trying to understand the charge carrier dynamics in a device but wants to remove the effects of a non-uniform charge generation generation profile.

- *From file*: This can be used to import generation profiles from a file. It us generally used to import the results of more complex optical simulations such as those from external FDTD solvers.

- *Constant value*: If you click on the arrow to the right of the simulation button you will be able to set the charge carrier generation rate within each layer by hand. Again this is generally used when trying to understand charge carrier dynamics or device performance without to consider a complex optical profile. This would allow one to plot graphs of charge generation rate v.s. V_{oc} etc..

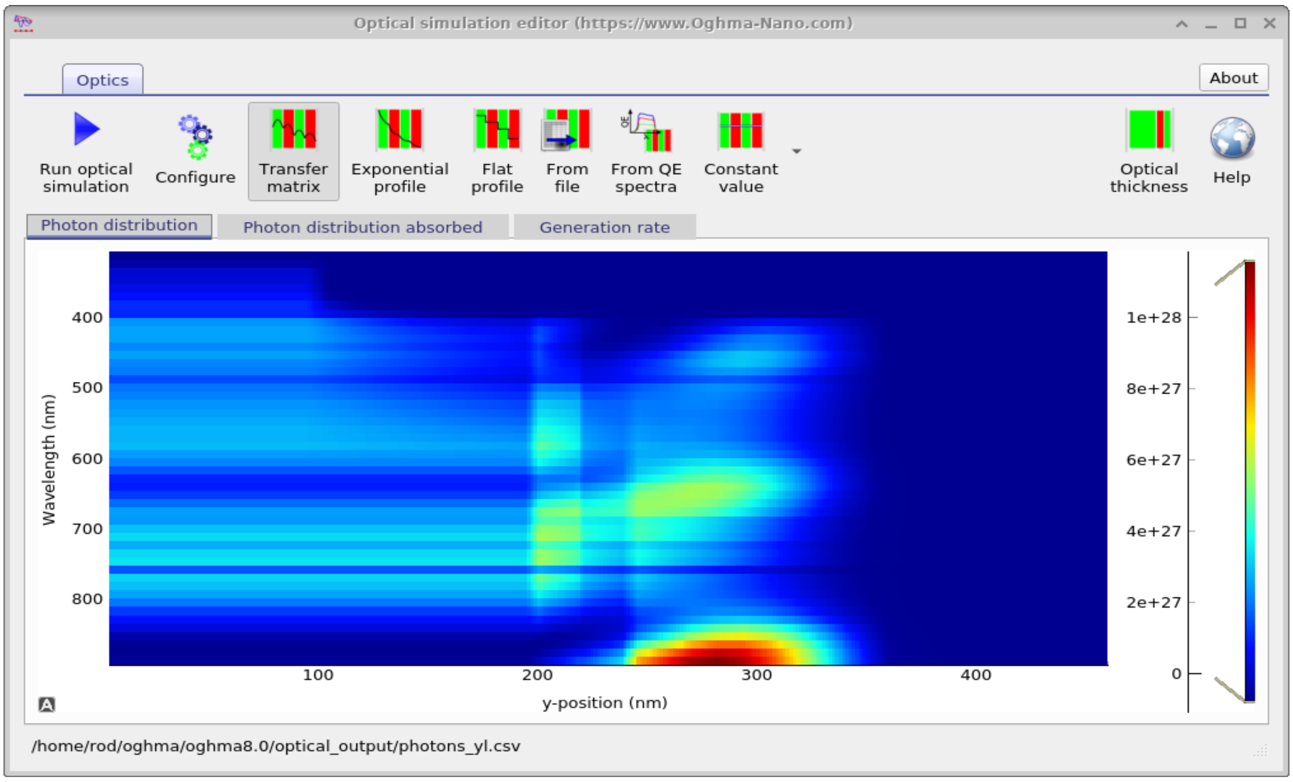
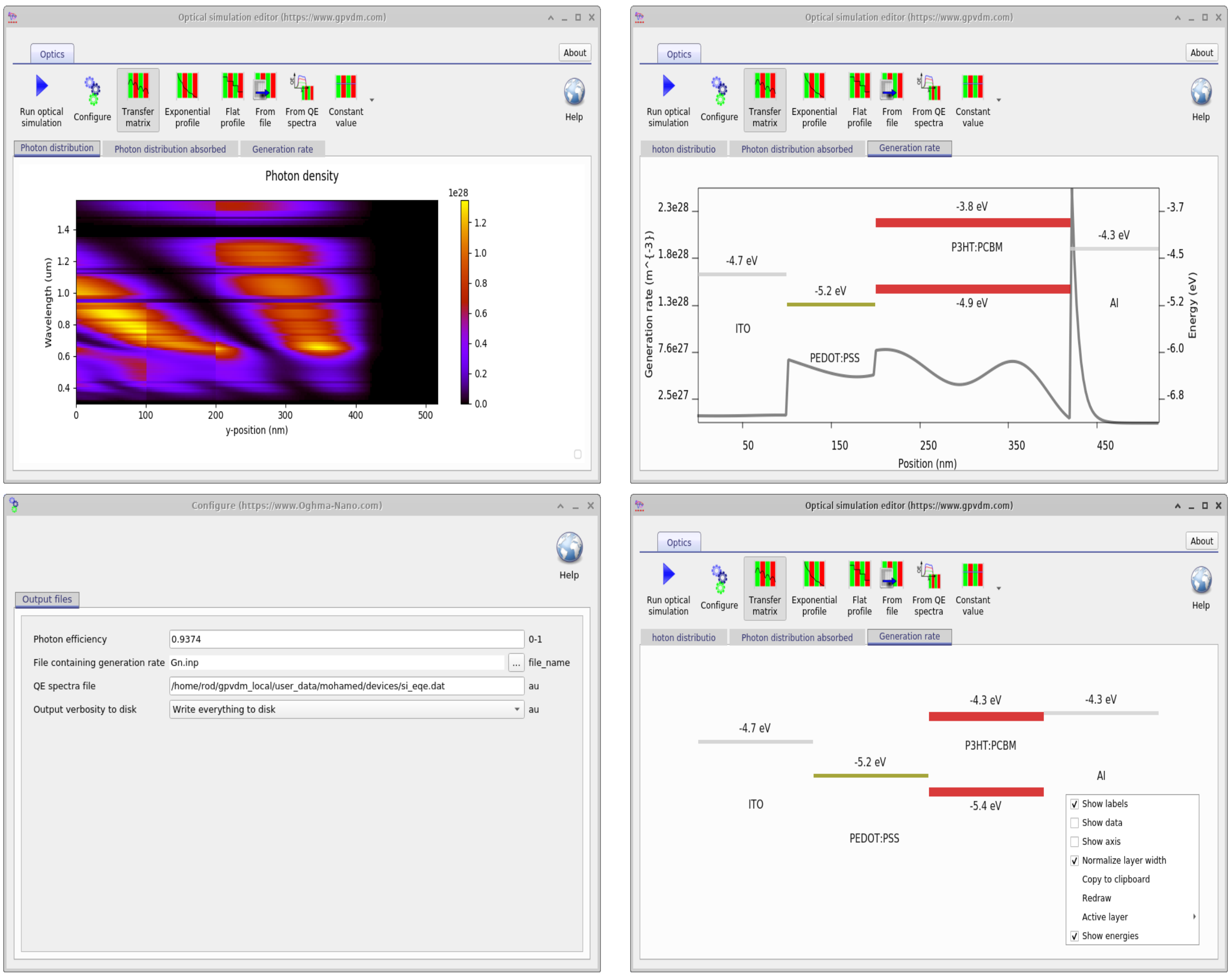


Figure 10.4: This is the Transfer matrix optical simulation window, here you can view the photon density and the rate of photon absorption in the device. The play button will run the simulation, exactly which flavour of transfer matrix simulation is run will depend on which push button you select from ribbon.

The optical simulation window has various tabs which can be used to explore how light interacts with the device. These can be seen in figure 10.5. The the top left image shows the photon density within the device, the image on the right shows the *total* photon density within the layers of the device. Notice how the reflection of the light of various layers causes interference patterns. The image on the bottom left shows the configuration of the optical model. A key parameter that can be seen in this window is the *Photon efficiency*, this parameter determines how many electron-hole pairs each photon that is absorbed within the active layer generates. In organic devices it accounts for geminate recombination, in other types of devices it should be set close to 1.0. Bottom right shows the same figure as in the top right of the figure, except by right clicking and playing with the menu options the figure has been converted into band diagram. This can be useful for generating band diagram figures for papers.



Photon efficiency	0.9374	0-1
File containing generation rate	Gn.inp	file_name
QE spectra file	/home/rod/gpvdm_local/user_data/mohamed/devices/si_eqe.dat	au
Output verbosity to disk	Write everything to disk	au

Figure 10.5: Various views of the optical simulation window

Running the transfer matrix simulation

As described above, the transfer matrix simulation can be run by directly clicking on the play button. However when running an electrical simulation the optical model is run automaticity without any user interaction. The only difference between the two ways of calling the optical simulation is that when the user directly calls the optical model more output files are written to disk, however when the optical simulation is run as part of an electrical simulation the number of files and output are limited as to not slow the electrical simulation.

10.2.2 Output files

After having run an optical simulation an overview of the results can be seen in the optical simulation window (Figure 10.4) as described above. However more in depth information can be obtained from the *Output* tab of the main window as shown in Figure 10.6.

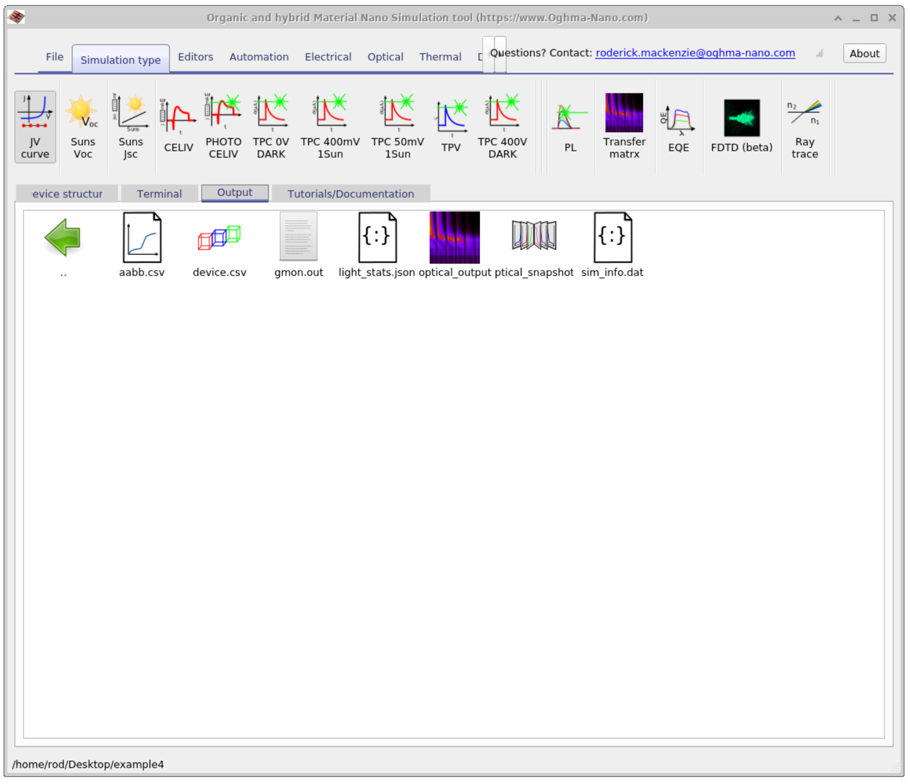


Figure 10.6: The output tab shows the output from the transfer matrix simulation. Usually the optical_output and optical_snapshots files are only generated when the optical simulation is run directly and not as part of another simulation.

In Figure 10.6 you can see two icons one called *Optical output* and the other called *optical_snapshots* (in the figure it reads "ptica_output" due to the text being hidden). If you double click on *Optical output* it will bring up figure 10.4. If you double click on *optical_snapshots* it will bring up Figure 10.7. The optical snapshot window allows the user to view photon density, absorbed photons and electric field of the light. The information is displayed per wavelength so a detailed overview of device performance can be gained.

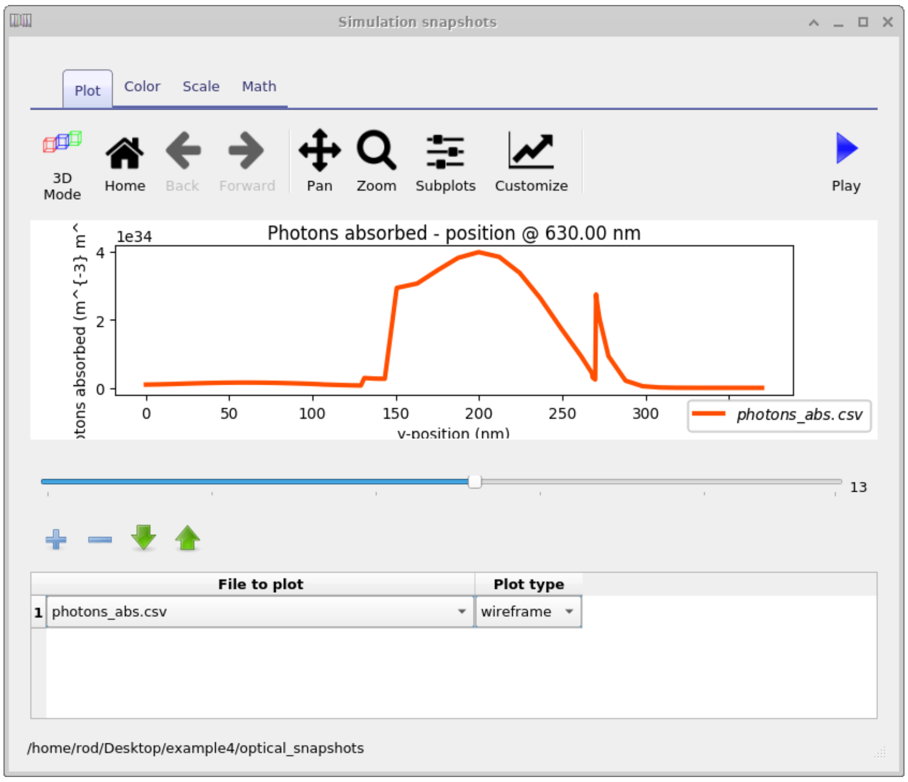


Figure 10.7: The optical snapshots window. This allows the user to view, Electric

The optical_snapshots directory in depth

The optical_snapshots folder was described above and when accessed through the graphical interface allows the user to access photon density, absorbed photons and electric field of the light as a function of wavelength. However, it is simply a normal directory and the user can access it through a file explorer. If you open the directory using a tool such as windows explorer you will see something comparable to what is shown on the left of Figure 10.8. You can see folders numbered from 0 to 12 each folder represents a simulated wavelength. If you open a directory, say number 0, you will be presented with the files shown in the right hand of figure 10.8. These files contain the following information:

File name	Description
alpha.csv	y-position v.s. absorption @ given wavelength
data.json	json file containing wavelength value
En.csv	y-position (m) v.s. Electric field with -ve component (V/m) @ given wavelength
Ep.csv	y-position (m) v.s. Electric field with +ve component (V/m) @ given wavelength
G.csv	y-position (m) v.s. Generation rate ($m^{-3}s^{-1}$)
n.csv	y-position (m) v.s. Real part of the refractive index n (au) @ given wavelength
photons.csv	y-position (m) v.s. photon density (m^{-3}) @ given wavelength
photons_abs.csv	y-position (m) v.s. photons absorbed ($m^{-3}s^{-1}$) @ given wavelength

Table 10.1: Files produced by the JV simulation

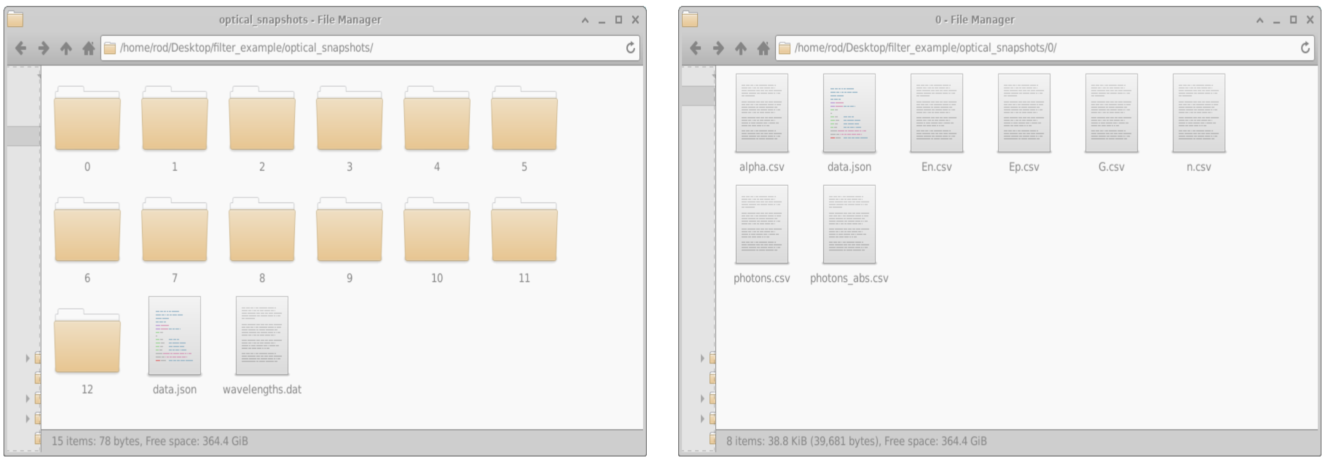
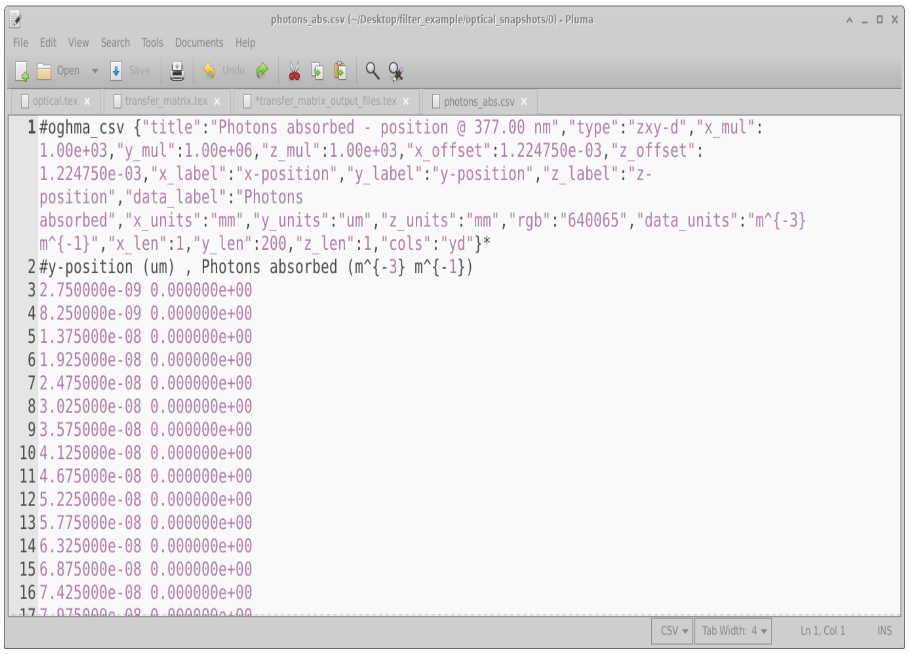

optical_snapshots - File Manager
/home/rod/Desktop/filter_example/optical_snapshots/
0
1
2
3
4
5
6
7
8
9
10
11
12
data.json
wavelengths.dat
15 items: 78 bytes, Free space: 364.4 GiB
0 - File Manager
/home/rod/Desktop/filter_example/optical_snapshots/0/
alpha.csv
data.json
En.csv
Ep.csv
G.csv
n.csv
photons.csv
photons_abs.csv
8 items: 38.8 KiB (39,681 bytes), Free space: 364.4 GiB

Figure 10.8: Various views of the optical simulation window

These files are simply plain text, if you open one it will look like 10.9. The first line of this file contains some information about the content of the file to help with plotting. The second line tells the user what the x and y axis contain then the following lines contain the data.



```
#oghma_csv {"title":"Photons absorbed - position @ 377.00 nm","type":"zxy-d","x_mul":
1.00e+03,"y_mul":1.00e+06,"z_mul":1.00e+03,"x_offset":1.224750e-03,"z_offset":
1.224750e-03,"x_label":"x-position","y_label":"y-position","z_label":"z-
position","data_label":"Photons
absorbed","x_units":"mm","y_units":"um","z_units":"mm","rgb":"640065","data_units":"m^{-3}
m^{-1}","x_len":1,"y_len":200,"z_len":1,"cols":"yd"}*
#y-position (um) , Photons absorbed (m^{-3} m^{-1})
2.750000e-09 0.000000e+00
8.250000e-09 0.000000e+00
1.375000e-08 0.000000e+00
1.925000e-08 0.000000e+00
2.475000e-08 0.000000e+00
3.025000e-08 0.000000e+00
3.575000e-08 0.000000e+00
4.125000e-08 0.000000e+00
4.675000e-08 0.000000e+00
5.225000e-08 0.000000e+00
5.775000e-08 0.000000e+00
6.325000e-08 0.000000e+00
6.875000e-08 0.000000e+00
7.425000e-08 0.000000e+00
```

Figure 10.9: The content of photons_abs.csv.

The optical_output folder in depth

While the optical_snapshots directory allows data to be plotted per simulated wavelength, the optical_output gives 2D maps of wavelength v.s. position for various simulation parameters. The content of the directory is shown in figure 10.10.

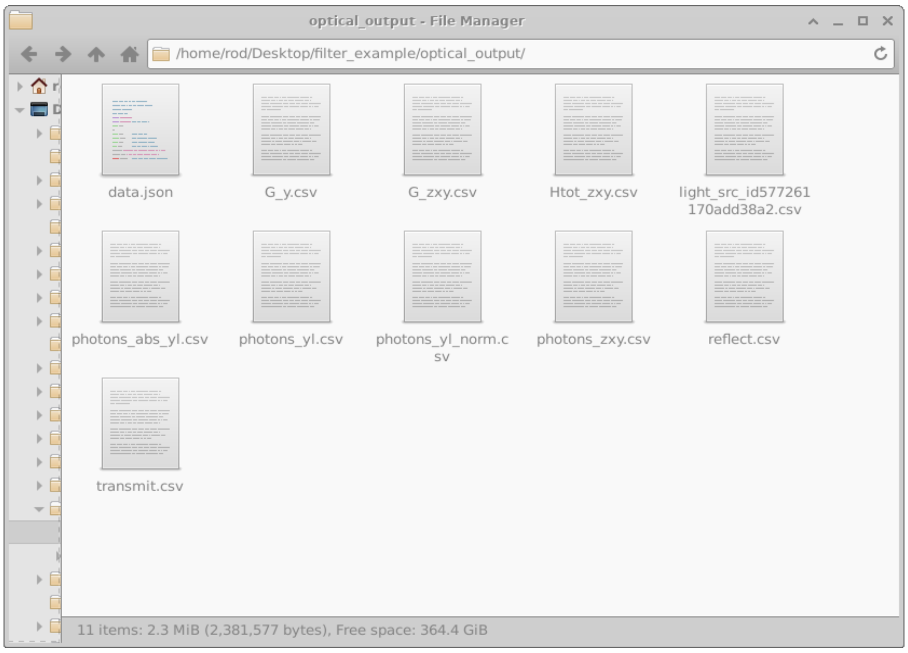

optical_output - File Manager
/home/rod/Desktop/filter_example/optical_output/
data.json
G_y.csv
G_zxy.csv
Htot_zxy.csv
light_src_id577261170add38a2.csv
photons_abs_yl.csv
photons_yl.csv
photons_yl_norm.csv
photons_zxy.csv
reflect.csv
transmit.csv
11 items: 2.3 MiB (2,381,577 bytes), Free space: 364.4 GiB

Figure 10.10: The content of optical_output directory

The files shown in Figure 10.10 are described in table 10.2.

File name	Description	Plot type
data.json	Miscellaneous simulation information in json format	1D
G_y.csv	y-position (m) v.s. Charge generation rate ($m^{-3}s^{-1}$)	2D
G_zxy.csv	zxy-position (m) v.s. Charge generation rate ($m^{-3}s^{-1}$)	2D
Htot_zxy.csv	zxy-position (m) v.s. Optical heat generation (Wm^{-3})	2D
light_src_id_xxx.csv	wavelength (m) v.s. Light intensity from source (W/m)	2D
photons_abs_yl.csv	wavelength (m) v.s. y-position (m) v.s. photons absorbed ($m^{-3}s^{-1}$)	2D
photons_yl.csv	wavelength (m) v.s. y-position (m) v.s. Photon density ($m^{-3}s^{-1}$)	2D
photons_yl_norm.csv	wavelength (m) v.s. y-position (m) v.s. Normalized photons (au)	2D
reflect.csv	wavelength (m) v.s. Light reflected from the stack	1D
transmit.csv	wavelength (m) v.s. Transmitted though the stack	1D

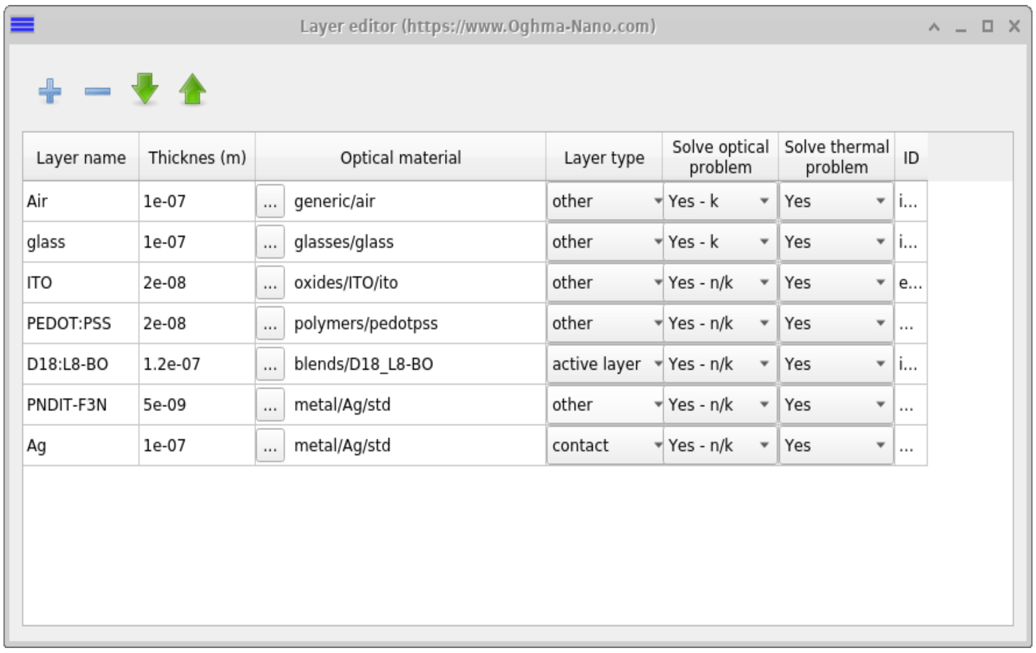
Table 10.2: Files produced by the JV simulation

10.2.3 Simulating optically thick layers (incoherent layers)

Typical optoelectronic devices have layer thicknesses between 10 nm and 100 nm. However often these devices are deposited on top of substrates that are between 10 mm and 1 cm thick and often one wants to not only simulate the device but also the impact of the substrate. Thus to perform this type of simulation one will need a simulation tool that covers length scales from the nm top the meter scale. There are three problems with doing this:

- Problem 1: *Simulating different length scales*: Computers don't like doing maths with numbers that are very big and small very often this results in large computational/rounding errors, there is more on this in secton 9.11.1.
- Problem 2: *The wavelength of light*: The wavelength of light is far smaller than 1 cm thus to get sensible answers out of the simulation one will need a very large number of mesh points correctly represent the constructive/negative interference of the light within the layer.
- Problem 3: *The light will not be coherent*: The transfer matrix model assumes light comes from a single direction at 90 degrees to the interface and there are no defects in the material. For a thick layer this will not be true.

To get around these issues OghmaNano uses two strategies. The first is to give the user the option to only consider absorption and neglect phase changes within a layer, to form a so called incoherent layer (this solves problem 2 and 3). This can be selected from the layer editor in the main window see Figure 10.11. Notice in the column entitled *Solve optical problem*, the first two layers (air and glass) have *Yes - k* selected, and the other layers have *Yes - n/k* selected. This means that in layers with *Yes - n/k* phase changes of the light will be considered but in the layers marked *Yes - k* only attenuation losses will be accounted for and thus it can be thought of as an incoherent layer.


Layer editor (https://www.Oghma-Nano.com)

Layer name	Thicknes (m)	Optical material	Layer type	Solve optical problem	Solve thermal problem	ID
Air	1e-07	... generic/air	other	Yes - k	Yes	i...
glass	1e-07	... glasses/glass	other	Yes - k	Yes	i...
ITO	2e-08	... oxides/ITO/ito	other	Yes - n/k	Yes	e...
PEDOT:PSS	2e-08	... polymers/pedotpss	other	Yes - n/k	Yes	...
D18:L8-BO	1.2e-07	... blends/D18_L8-BO	active layer	Yes - n/k	Yes	i...
PNDIT-F3N	5e-09	... metal/Ag/std	other	Yes - n/k	Yes	...
Ag	1e-07	... metal/Ag/std	contact	Yes - n/k	Yes	...

Figure 10.11: The layer editor showing both *coherent layers* and *incoherent layers*

To get around the problem of having to simulate different length scales (problem 1) OghmaNano allows the user to set an *effective* optical depth for any layer. So one can for example setup a layer in the layer editor of width 100 nm, but set it's *effective* depth to a much larger value such as 1 m. This works by multiplying the absorption coefficient of the layer by the ratio:

$$\alpha_{effective}(\lambda) = \alpha(\lambda)\frac{L_{effective}}{L_{simulation}} \tag{10.4}$$

Where $\alpha_{effective}$ is the effective absorption used in the simulation, α is the true value of absorption for the material, $L_{effective}$ is the effective layer thickness (say 1 m or 1 km), and $L_{simulation}$ is the thickness of the layer in the simulation window. Not only does this approach reduce numerical issues (problem 1) but it also allows the user to plot meaningful graphs of the simulation results, without most of the plot taken up by the substrate and the device only appearing as a tiny slither on the edge of the graph.

If you want to use this feature, then set up a device structure as shown in 10.11 then in the transfer matrix window click the Optical Thickness button (top right of Figure 10.12). Then the window 10.13 will appear. In this window you can set the *effective optical thickness* of any layer. In this case we have set the glass to be 1 meter thick.

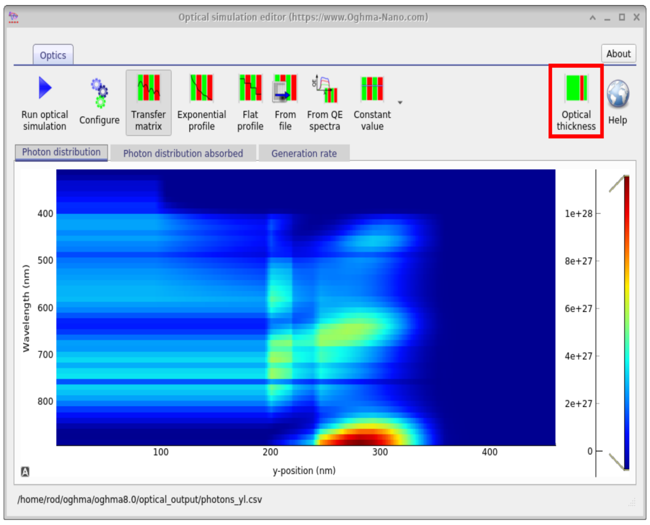
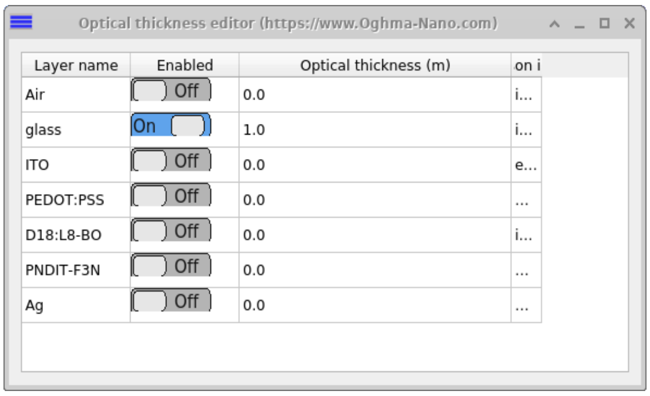


Figure 10.12: Opening the effective optical thickness window.



Layer name	Enabled	Optical thickness (m)	on i
Air	Off	0.0	i...
glass	On	1.0	i...
ITO	Off	0.0	e...
PEDOT:PSS	Off	0.0	...
D18:L8-BO	Off	0.0	i...
PNDIT-F3N	Off	0.0	...
Ag	Off	0.0	...

Figure 10.13: The effective optical thickness window, you can see that for this structure the optical thickness of glass has been set to 1 meter.

10.2.4 Theory of the transfer matrix method

On the left of the interface the electric field is given by

$$E_1 = E_1^+ e^{-jk_1 z} + E_1^- e^{jk_1 z} \tag{10.5}$$

and on the right hand side of the interface the electric field is given by

$$E_2 = E_2^+ e^{-jk_2 z} + E_2^- e^{jk_2 z} \tag{10.6}$$

Maxwel's equations give us the relationship between the electric and magnetic fields for a plane wave.

$$\nabla \times E = -j\omega\mu H \tag{10.7}$$

which simplifies to:

$$\frac{\partial E}{\partial z} = -j\omega\mu H \tag{10.8}$$

Applying equation 10.8 to equations 10.5-10.6, we can get the magnetic field on the left of the interface

$$-j\mu\omega H_1^y = -jk_1 E_1^+ e^{-jk_1 z} + jk_1 E_1^- e^{jk_1 z} \tag{10.9}$$

and on the right of the interface

$$-j\mu\omega H_2^y = -jk_2 E_2^+ e^{-jk_2 z} + jk_2 E_2^- e^{jk_2 z}. \tag{10.10}$$

Tidying up gives,

$$H_1^y = \frac{k}{\omega\mu} E_1^+ e^{-jk_1 z} - \frac{k}{\omega\mu} E_1^- e^{jk_1 z} \tag{10.11}$$

$$H_2^y = \frac{k}{\omega\mu} E_2^+ e^{-jk_2 z} - \frac{k}{\omega\mu} E_2^- e^{jk_2 z} \tag{10.12}$$

Boundary conditions

We now apply the electric and magnetic boundary conditions[18]

$$\mathbf{n} \times (\mathbf{E_2} - \mathbf{E_1}) = 0 \tag{10.13}$$

$$\mathbf{n} \times (\mathbf{H_2} - \mathbf{H_1}) = 0 \tag{10.14}$$

We let the interface be at z=0, which gives,

$$(E_2^+ + E_2^-) - (E_1^+ + E_1^-) = 0 \tag{10.15}$$

and

$$\frac{k_1}{\omega\mu}(E_2^+ - E_2^-) - (E_1^+ - E_1^-)\frac{k_2}{\omega\mu} = 0 \tag{10.16}$$

. The wavevector is given by

$$k = \frac{2\omega}{\lambda} = \frac{\omega n}{c} \tag{10.17}$$

. We can therefore write the magnetic boundary condition as

$$n_2(E_2^+ - E_2^-) - n_1(E_1^+ - E_1^-) = 0 \tag{10.18}$$

Forward propagating wave

Rearrange equation, 10.18 to give,

$$E_1^- = E_1^+ - \frac{n_2}{n_1}(E_2^+ - E_2^-) \tag{10.19}$$

Inserting in equation 10.15, gives

$$E_2^+ + E_2^- = E_1^+ + E_1^+ - \frac{n_2}{n_1}(E_2^+ - E_2^-) \tag{10.20}$$

$$2E_1^+ = E_2^+ + E_2^- + \frac{n_2}{n_1}(E_2^+ - E_2^-) \tag{10.21}$$

$$2E_1^+ \frac{n_1}{n_1+n_2} = E_2^+ + E_2^- \frac{n_1-n_2}{n_1+n_2} \tag{10.22}$$

Backwards propagating wave

Rearrange equation, 10.18 to give,

$$E_1^+ = E_1^- + \frac{n_2}{n_1}(E_2^+ - E_2^-) \tag{10.23}$$

Inserting in equation 10.15, gives

$$E_2^+ + E_2^- = E_1^- + \frac{n_2}{n_1}(E_2^+ - E_2^-) + E_1^- \tag{10.24}$$

$$2E_1^- = E_2^+ + E_2^- - \frac{n_2}{n_1}(E_2^+ - E_2^-) \tag{10.25}$$

$$2E_1^- \frac{n_1}{n_1+n_2} = E_2^+ \frac{n_1-n_2}{n_1+n_2} + E_2^- \tag{10.26}$$

Which is the same result as obtained in [19].

These equations become:

$$E_1^- t_{12} = E_2^+ r_{12} + E_2^- \tag{10.27}$$

and

$$E_1^+ t_{12} = E_2^+ + E_2^- r_{12} \tag{10.28}$$

Accounting for propagation we can write. Note the change in sign between [19] and this work, this is because of how I have defined my wave equation.

$$E_1^+ t_{12} = E_2^+ e^{\zeta_2 d_1} + E_2^- r_{12} e^{-\zeta_2 d_1} \tag{10.29}$$

and

$$E_1^- t_{12} = E_2^+ r_{12} e^{\zeta_2 d_1} + E_2^- e^{-\zeta_2 d_1} \tag{10.30}$$

where

$$\zeta = \frac{2\pi}{\lambda}\bar{n} \tag{10.31}$$

For a device with non reflecting back contacts:

$$\begin{pmatrix} e^{\zeta d} & 0 & 0 & 0 & r_{01}e^{-\zeta d} & 0 & 0 & 0 \\ -t_{12} & e^{\zeta d} & 0 & 0 & 0 & r_{12}e^{-\zeta d} & 0 & 0 \\ 0 & -t_{23} & e^{\zeta d} & 0 & 0 & 0 & r_{23}e^{-\zeta d} & 0 \\ 0 & 0 & -t_{34} & e^{\zeta d} & 0 & 0 & 0 & r_{34}e^{-\zeta d} \\ 0 & r_{12}e^{\zeta d_1} & 0 & 0 & -t_{12} & e^{-\zeta d} & 0 & 0 \\ 0 & 0 & r_{23}e^{\zeta d} & 0 & 0 & -t_{23} & e^{-\zeta d} & 0 \\ 0 & 0 & 0 & r_{34}e^{\zeta d} & 0 & 0 & -t_{34} & e^{-\zeta d} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -t_{45} \end{pmatrix} \begin{pmatrix} E_1^+ \\ E_2^+ \\ E_3^+ \\ E_4^+ \\ E_1^- \\ E_2^- \\ E_3^- \\ E_4^- \end{pmatrix} = \begin{pmatrix} t_{01}E_{external} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

10.2.5 Refractive index and absorption

$$E(z,t) = Re(E_0 e^{j(-kz+\omega t)}) = Re(E_0 e^{j(\frac{-2\pi(n+j\kappa)}{\lambda}z+\omega t)}) = e^{\frac{2\pi\kappa z}{\lambda}} Re(E_0 e^{\frac{j(-2\pi(n+j\kappa)}{\lambda}z+\omega t}) \tag{10.32}$$

And because the intensity is proportional to the square of the electric field the absorption coefficient becomes

$$e^{-\alpha x} = e^{\frac{2\pi\kappa z}{\lambda}} \tag{10.33}$$

$$\alpha = -\frac{4\pi\kappa}{\lambda_0} \tag{10.34}$$

10.3 Optical mode solve 1D/2D

Related YouTube videos:

Tutorial on calculating TE/TM modes in slab waveguides

Text to be added later.

10.4 Finite Difference Time Domain

Finite Difference Time Domain (FDTD) is a very computationally intensive way to simulate the propagation of electromagnetic radiation. It solves full on Maxwell's equations in time domain making no assumptions. This approach has only been possible in the last few years with the increase in computing power.

Food for thought: Before using FDTD, you should be aware that in many cases when simulating a device, using FDTD is like using a sledge hammer to crack a nut. For example when simulating a standard solar cell, one could use FDTD, this would involve simulating a wave front entering the cell through the top contact in time domain and waiting until the optical field reaches steady state then calculating how much light is being absorbed. This would require thousands of time domain simulation steps per wavelength. However usually one is not interested in the time evolution of light in a solar cell as sunlight varies very slowly indeed, so one is better off not using a steady state method but other methods which assume light has already reached steady state, such as the transfer matrix method discussed above.

Never the less, FDTD is an import method, and can be used to design and understand complex devices.

Related YouTube videos:

Generating photonic crystal structures for FDTD simulation

Tutorial on simulating photonic crystal waveguides using FDTD

10.4.1 Running an FDTD simulation

There is a demo FDTD simulation in the new simulations window. To open this click on new simulation in the file ribbon of the main window. The window in figure 10.14 will appear. Double click on the "Photonic-xtal FDTD" simulation to open it. Save the simulation on your local hard disk, don't save it on a remote disk, USB disk or OneDrive they will be too slow to run the simulation.

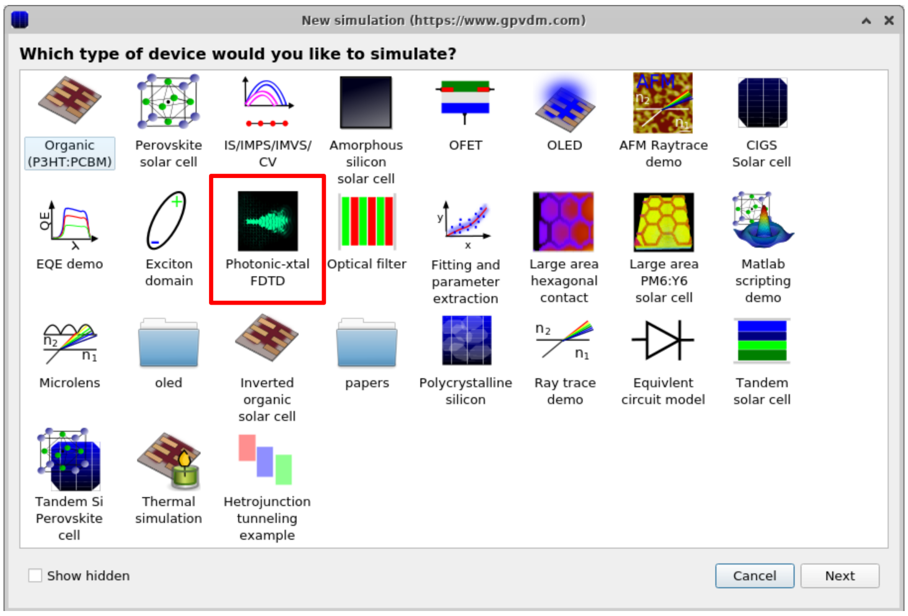
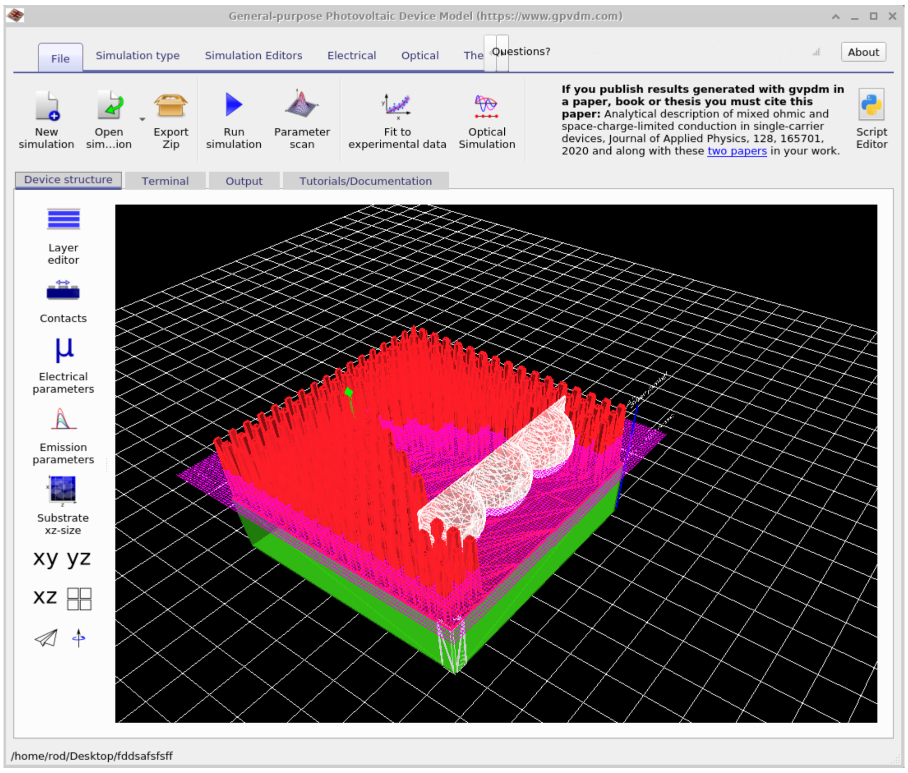


Figure 10.14: The new simulation window

Once you have opened the simulation you should get a window which appears figure 10.15. If you click on the play button the FDTD simulation will run, it will take around 30 seconds to run.



If you publish results generated with gvpdm in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

Figure 10.15: The initial FDTD simulation window. Use the slider to look at the results in time domain, and the drop down menu to select which field you are going to look at.

After the simulation has run click on the *Output* tab and 10.16 you will see the FDTD *snapshots* folder, this can be seen in figure 10.16. If you double click on this the FDTD snapshots window will appear which is shown in figure 10.17. This window will allow you to step through the simulations. If you click in the files to plot box, you will be able to select which field you plot. You will be able to select the *Ey*, *Ex* or *Ez* fields. In this case select the Ey field. Then use the slider bar to step through the field as a function of time.

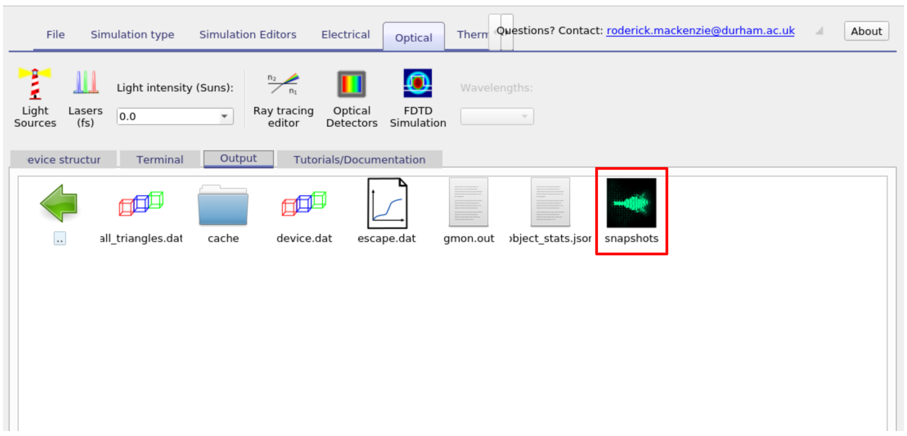


Figure 10.16: The output tab after having run an FDTD simulation, the key output is the Snapshots folder where the fields are stored.

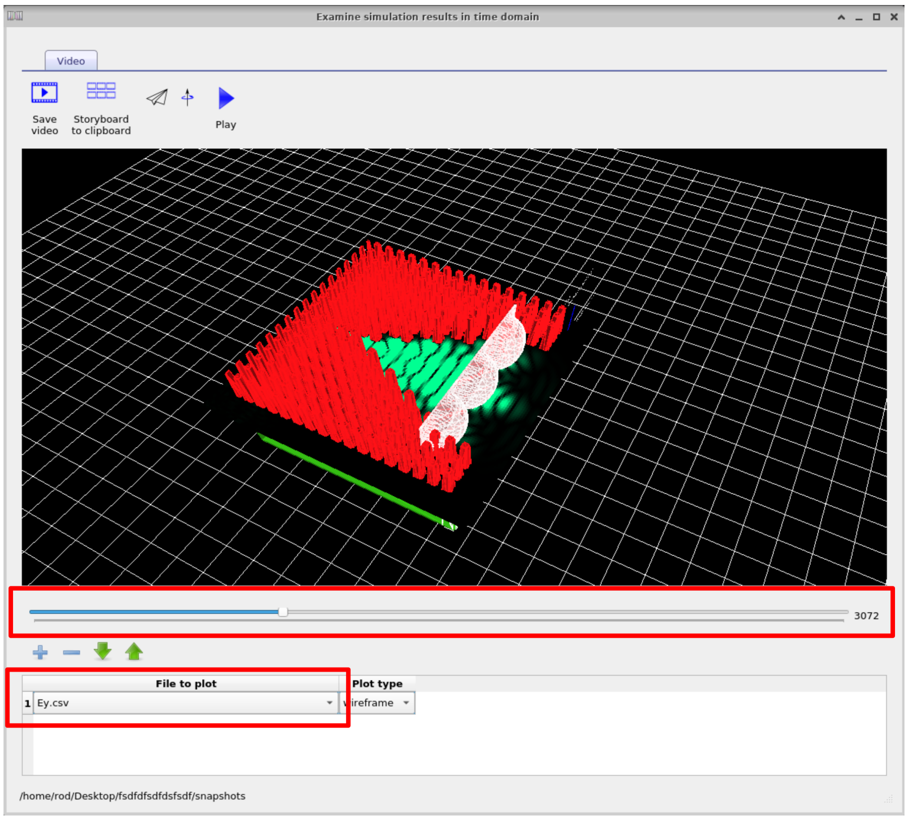

Examine simulation results in time domain
Video
Save video
Storyboard to clipboard
Play
3072
File to plot
Plot type
1
Ey.csv
Wireframe
/home/rod/Desktop/fsdfdfsdfdsfsdf/snapshots

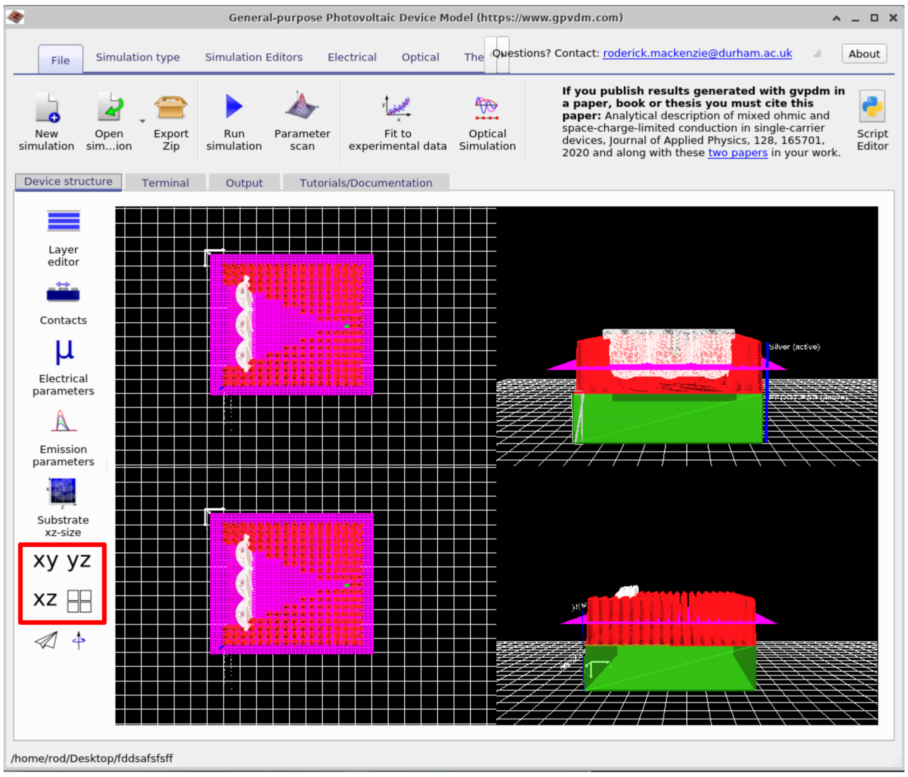
Figure 10.17: The FDTD snapshots window.

10.4.2 Manipulating objects in OghmaNano

Now close the snapshot viewer and go back to the main simulation window and select the *Device tab*. On the left of the window, you will see four buttons *xy*, *yz*, *xz* and for little square boxes this can be seen in figure 10.18. Try clicking them to see what happens to the view of the device. After you have had a play select the *xz* option, so that the screen looks like the left hand side of 10.19. If you left click on the lenses, you will notice that you will be able to move them around. Try to move the lenses back in the device so that your device looks more like the right hand side of figure 10.18. If you hold shift down while dragging an object you can rotate it on the spot.

If you right click on the lenses and select *Edit* you will be able to bring up the object Editor. This shows the user all the object properties, this is visible in figure 10.20. Try changing the object type from *convex_lens* to *concave_lens* by clicking the edit button and rerunning the simulation. If you want to add your own shapes to the shape data base see section 15.2. Using this window you can also change the material which is used in the FDTD simulation, the color of the object as well as its position or rotational angle.

The *shape enabled* button enables you to turn off the shape if you don't want it in the simulation. If this shape were also electrically active you could also use this window to configure the electrical parameters.



If you publish results generated with gvpdm in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

Figure 10.18: Changing the object view in OghmaNano

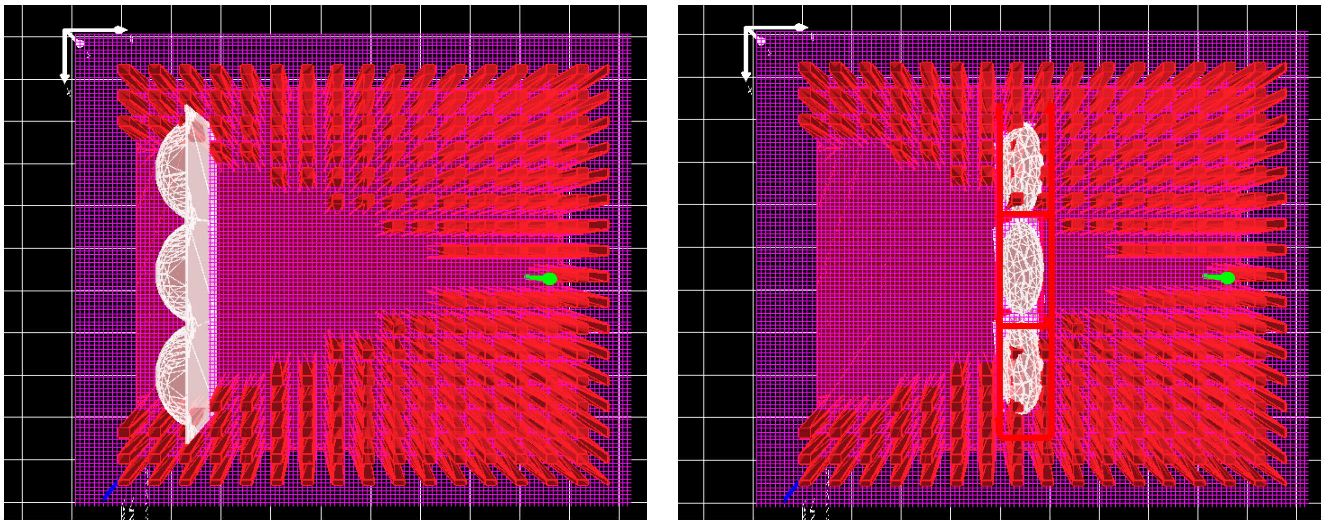
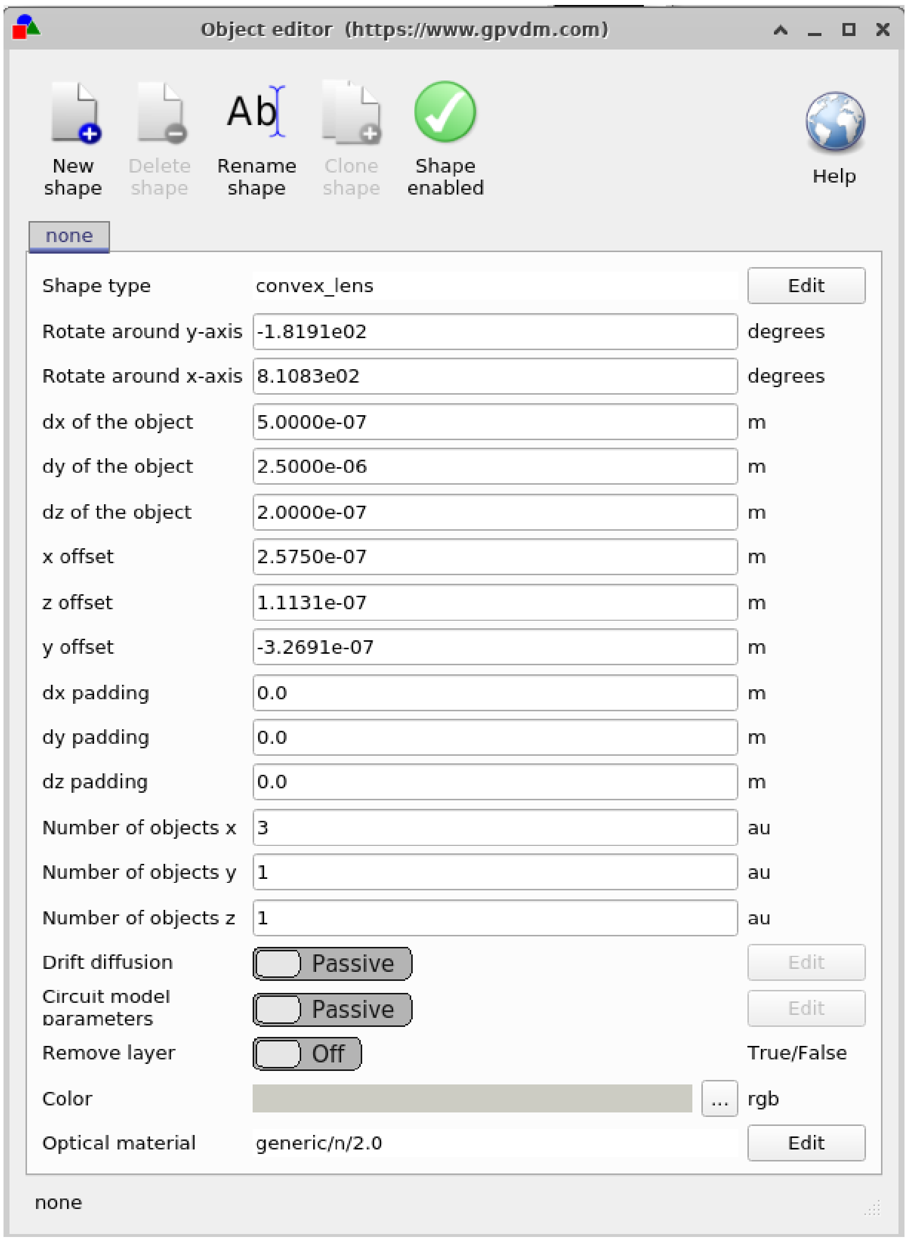


Figure 10.19: An example of moving objects in the simulation window.



Shape type	convex_lens	Edit
Rotate around y-axis	-1.8191e02	degrees
Rotate around x-axis	8.1083e02	degrees
dx of the object	5.0000e-07	m
dy of the object	2.5000e-06	m
dz of the object	2.0000e-07	m
x offset	2.5750e-07	m
z offset	1.1131e-07	m
y offset	-3.2691e-07	m
dx padding	0.0	m
dy padding	0.0	m
dz padding	0.0	m
Number of objects x	3	au
Number of objects y	1	au
Number of objects z	1	au
Drift diffusion	Passive	Edit
Circuit model parameters	Passive	Edit
Remove layer	Off	True/False
Color	...	rgb
Optical material	generic/n/2.0	Edit

none

Figure 10.20: The object viewer. This window is brought up by right clicking on an object and selecting *Edit*.

10.4.3 Manipulating light sources in OghmaNano

Also visible in figure 10.19 is the light source given by the green arrow. Try moving this more forward in the device.

10.4.4 Theoretical background

References for this section are [20]. This section of the manual aims to describe the FDTD code in full with verbose derivations to help understanding/pick up errors.

Ampere's law is given as [20]

$$\sigma \boldsymbol{E} + \epsilon \frac{\partial \boldsymbol{E}}{\partial t} = \nabla \times \boldsymbol{H} = \begin{vmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ H_x & H_y & H_z \end{vmatrix} \tag{10.35}$$

which can be expanded as

$$\sigma E_x + \epsilon \frac{\partial E_x}{\partial t} = \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \tag{10.36}$$

$$\sigma E_y + \epsilon \frac{\partial E_y}{\partial t} = -\frac{\partial H_z}{\partial x} + \frac{\partial H_x}{\partial z} \tag{10.37}$$

$$\sigma E_z + \epsilon \frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \tag{10.38}$$

For the case $\frac{\partial}{\partial y} = 0$

$$\begin{aligned} \sigma E_x + \epsilon \frac{\partial E_x}{\partial t} &= -\frac{\partial H_y}{\partial z} \\ \sigma E_y + \epsilon \frac{\partial E_y}{\partial t} &= -\frac{\partial H_z}{\partial x} + \frac{\partial H_x}{\partial z} \\ \sigma E_z + \epsilon \frac{\partial E_z}{\partial t} &= \frac{\partial H_y}{\partial x} \end{aligned} \tag{10.39}$$

for E_x

$$\begin{aligned} &\sigma E_x + \epsilon \frac{\partial E_x}{\partial t} = -\frac{\partial H_y}{\partial z} \\ &\sigma \frac{E_x^{t+1}[] + E_x^t[]}{2} + \epsilon \frac{E_x^{t+1}[] - E_x^t[]}{\Delta t} = -\frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} \\ &\sigma \frac{E_x^{t+1}[]}{2} + \epsilon \frac{E_x^{t+1}[]}{\Delta t} = -\frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} - \sigma \frac{E_x^t[]}{2} + \epsilon \frac{E_x^t[]}{\Delta t} \\ &\sigma \frac{E_x^{t+1}[]}{2} + \epsilon \frac{E_x^{t+1}[]}{\Delta t} = -\frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} - \sigma \frac{E_x^t[]}{2} + \epsilon \frac{E_x^t[]}{\Delta t} \\ &\frac{\sigma \Delta t + 2\epsilon}{2\Delta t} E_x^{t+1}[] = -\frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} - \sigma \frac{E_x^t[]}{2} + \epsilon \frac{E_x^t[]}{\Delta t} \\ &E_x^{t+1}[] = \left(-\frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} - \sigma \frac{E_x^t[]}{2} + \epsilon \frac{E_x^t[]}{\Delta t} \right) \frac{2\Delta t}{\sigma \Delta t + 2\epsilon} \end{aligned} \tag{10.40}$$

for E_y

$$\begin{aligned} &\sigma E_y + \epsilon \frac{\partial E_y}{\partial t} = -\frac{\partial H_z}{\partial x} + \frac{\partial H_x}{\partial z} \\ &\sigma \frac{E_y^{t+1}[] + E_y^t[]}{2} + \epsilon \frac{E_y^{t+1}[] - E_y^t[]}{\Delta t} = -\frac{H_z^{t+\frac{1}{2}}[\frac{1}{2}] - H_z^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x} + \frac{H_x^{t+\frac{1}{2}}[\frac{1}{2}] - H_x^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} \\ &\sigma \frac{E_y^{t+1}[]}{2} + \epsilon \frac{E_y^{t+1}[]}{\Delta t} = -\frac{H_z^{t+\frac{1}{2}}[\frac{1}{2}] - H_z^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x} + \frac{H_x^{t+\frac{1}{2}}[\frac{1}{2}] - H_x^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} - \sigma \frac{E_y^t[]}{2} + \epsilon \frac{E_y^t[]}{\Delta t} \\ &E_y^{t+1}[] = \left(-\frac{H_z^{t+\frac{1}{2}}[\frac{1}{2}] - H_z^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x} + \frac{H_x^{t+\frac{1}{2}}[\frac{1}{2}] - H_x^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z} - \sigma \frac{E_y^t[]}{2} + \epsilon \frac{E_y^t[]}{\Delta t} \right) \frac{2\Delta t}{\sigma \Delta t + 2\epsilon} \end{aligned} \tag{10.41}$$

for E_z

$$\begin{aligned}
&\sigma E_z + \epsilon\frac{\partial E_z}{\partial t} = \frac{\partial H_y}{\partial x}\\
&\sigma\frac{E_z^{t+1}[] + E_z^t[]}{2} + \epsilon\frac{E_z^{t+1}[] - E_z^t[]}{\Delta t} = \frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x}\\
&\sigma\frac{E_z^{t+1}[]}{2} + \epsilon\frac{E_z^{t+1}[]}{\Delta t} = \frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x} - \sigma\frac{E_z^t[]}{2} + \epsilon\frac{E_z^t[]}{\Delta t}\\
&E_z^{t+1}[] = \left(\frac{H_y^{t+\frac{1}{2}}[\frac{1}{2}] - H_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x} - \sigma\frac{E_z^t[]}{2} + \epsilon\frac{E_z^t[]}{\Delta t}\right)\frac{2\Delta t}{\sigma\Delta t + 2\epsilon}
\end{aligned} \tag{10.42}$$

Faraday's law is given as [20]

$$-\sigma_m \boldsymbol{H} - \mu\frac{\partial \boldsymbol{H}}{\partial t} = \nabla \times \boldsymbol{E} = \begin{vmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ E_x & E_y & E_z \end{vmatrix} \tag{10.43}$$

which can be expanded to give:

$$-\sigma_m H_x - \mu\frac{\partial H_x}{\partial t} = \frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} \tag{10.44}$$

$$-\sigma_m H_y - \mu\frac{\partial H_y}{\partial t} = -\frac{\partial E_z}{\partial x} + \frac{\partial E_x}{\partial z} \tag{10.45}$$

$$-\sigma_m H_z - \mu\frac{\partial H_z}{\partial t} = \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \tag{10.46}$$

With $\sigma_m = 0$ and $\frac{\partial}{\partial y} = 0$

$$\begin{aligned}
\frac{\partial H_x}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z}\right)\\
\frac{\partial H_y}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}\right)\\
\frac{\partial H_z}{\partial t} &= -\frac{1}{\mu}\left(\frac{\partial E_y}{\partial x}\right)
\end{aligned} \tag{10.47}$$

which discretizing gives

$$\begin{aligned}
H_x^{t+1} &= \frac{1}{\mu}\left(\frac{E_y^{t+\frac{1}{2}}[\frac{1}{2}] - E_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z}\right)\Delta t + H_x^t[]\\
H_y^{t+1} &= \frac{1}{\mu}\left(\frac{E_z^{t+\frac{1}{2}}[\frac{1}{2}] - E_z^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x} - \frac{E_x^{t+\frac{1}{2}}[\frac{1}{2}] - E_x^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta z}\right)\Delta t + H_y^t[]\\
H_z^{t+1} &= \frac{1}{\mu}\left(-\frac{E_y^{t+\frac{1}{2}}[\frac{1}{2}] - E_y^{t+\frac{1}{2}}[-\frac{1}{2}]}{\Delta x}\right)\Delta t + H_x^z[]
\end{aligned} \tag{10.48}$$

10.4.5 Ray tracing model

Add text.

Chapter 11

Simple circuit simulations

OghmaNano was primarally 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 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 connected to the diodes, thus enabling photocurrent to be correctly simulated. There are a few examples if circuit simulations in the mode, these can be found in the *Simple Diode Model* folder of the new simulation folder (see figure 11.1.).

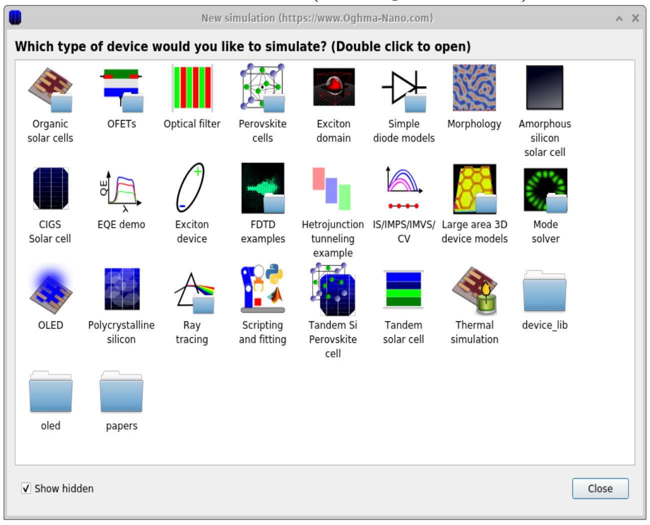


Figure 11.1: Selecting the *Simple circuit simulation example*

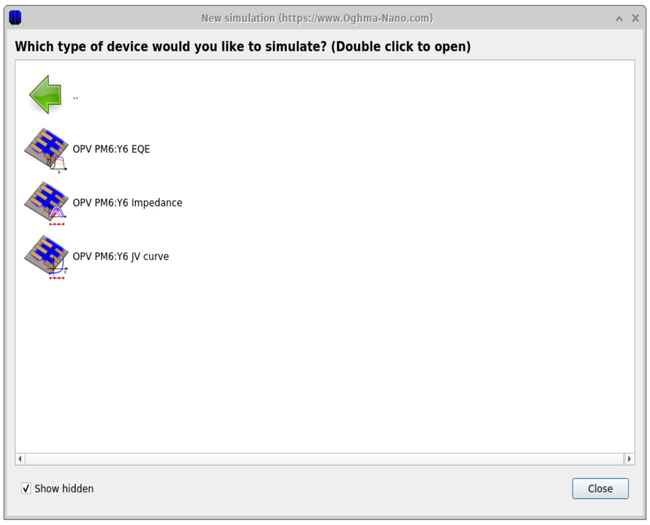

New simulation (https://www.Oghma-Nano.com)
Which type of device would you like to simulate? (Double click to open)
..
OPV PM6:Y6 EQE
OPV PM6:Y6 Impedance
OPV PM6:Y6 JV curve
Show hidden
Close

Figure 11.2: Selecting the example that generates the JV curve

With in this folder there are a few example circuit simulations (see Figure 11.2).

If one opens the *OPV PM6:Y6 JV curve* one will get a simulation that looks just like other simulations in OghmaNano (see Figure 11.2), 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 11.4. 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 $i(t,V) = \frac{I_0 * V}{V_0 + d}^m$

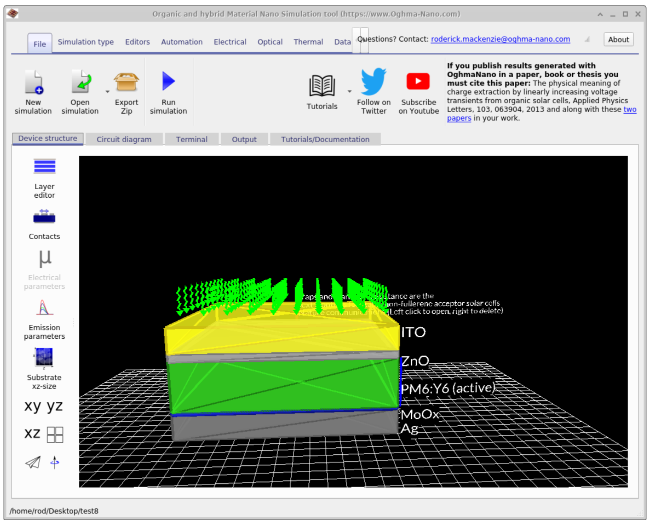
- Wire: A perfect wire with no parasitic parameters.

- 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.

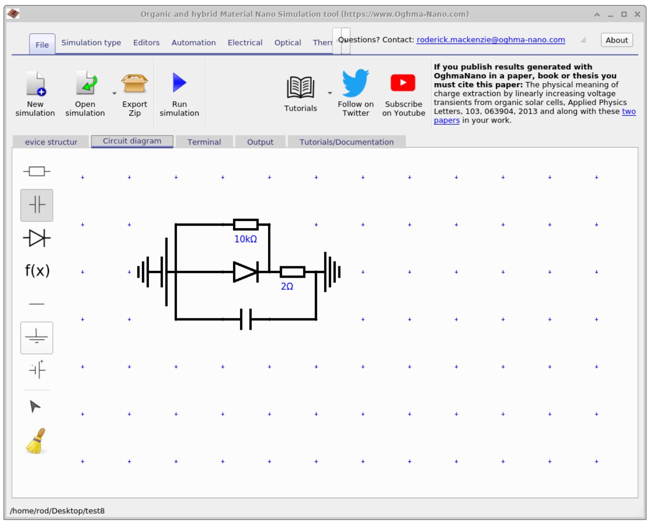
- Pointer: This ise used to select and edit circuit elements.

- Brush: This is used to delete circuit elements.



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: The physical meaning of charge extraction by linearly increasing voltage transients from organic solar cells, Applied Physics Letters, 103, 063904, 2013 and along with these two papers in your work.

Figure 11.3: The usual interface opens when the circuit simulation is selected.



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: The physical meaning of charge extraction by linearly increasing voltage transients from organic solar cells, Applied Physics Letters, 103, 063904, 2013 and along with these two papers in your work.

Figure 11.4: 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 11.5, and zoomed in Figure 11.6. Figure 11.6 shows the configuration window for the diode component. From the top the options are:

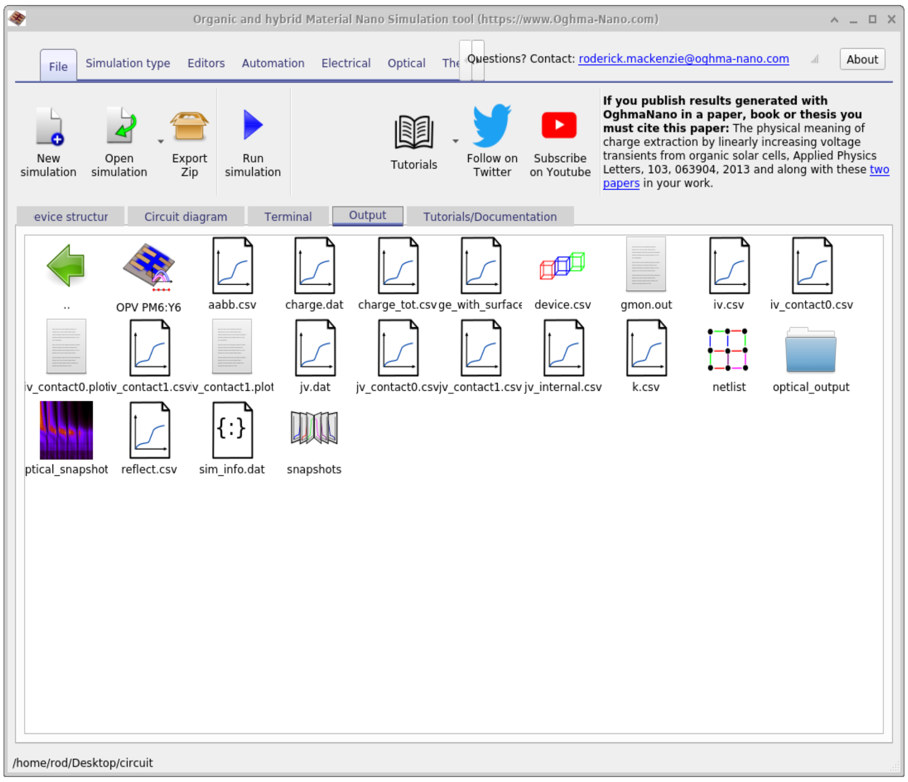
- Component: This can be used to change what component the circuit element represents.

- 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.



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: The physical meaning of charge extraction by linearly increasing voltage transients from organic solar cells, Applied Physics Letters, 103, 063904, 2013 and along with these two papers in your work.

Figure 11.7: The output of circuit simulation window is exactly as it would be for standard drift diffusion simulations.

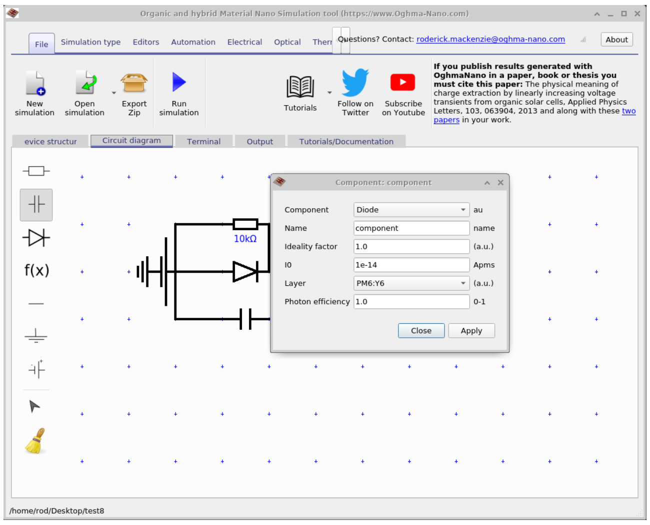
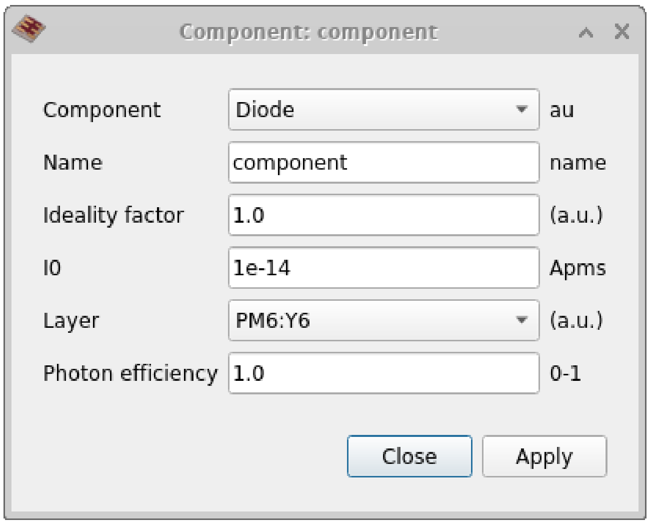


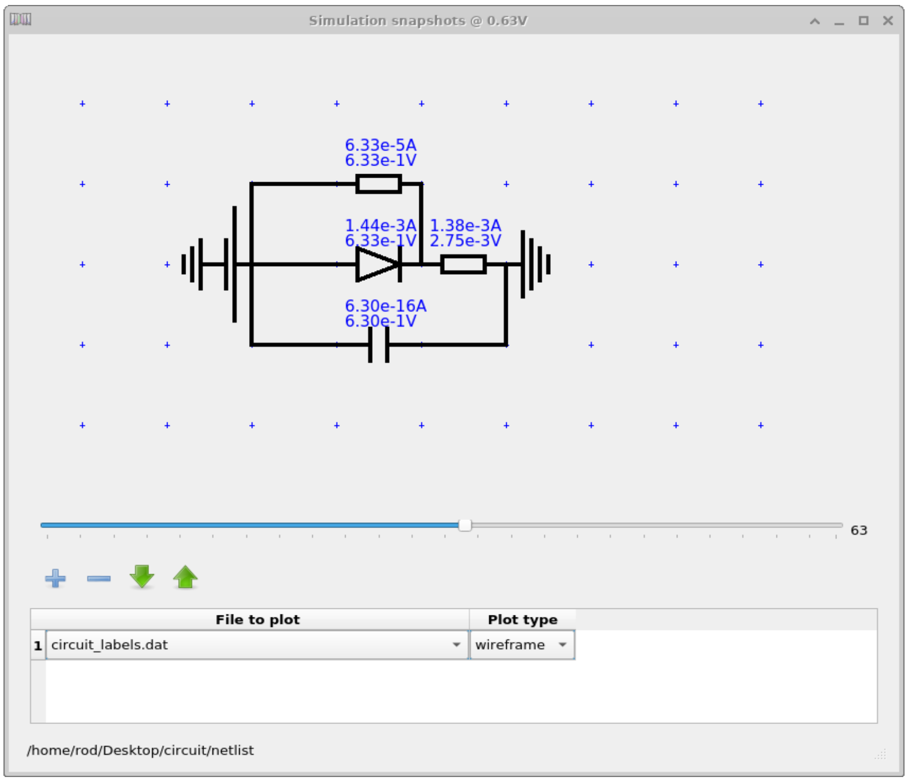
Figure 11.5: Editing the component values of a diode.



Component	Diode	au
Name	component	name
Ideality factor	1.0	(a.u.)
I0	1e-14	Apms
Layer	PM6:Y6	(a.u.)
Photon efficiency	1.0	0-1

Figure 11.6: 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 11.8, when you double click on this it brings up the Net list window which is visible in Figure 11.8, 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.



File to plot	Plot type
1 circuit_labels.dat	wireframe

/home/rod/Desktop/circuit/netlist

Figure 11.8: The net list, showing the voltages over components and currents through components.

11.0.1 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 11.9), the result of which can be seen in Figure 11.10 where the file real_imag.csv has been plotted.

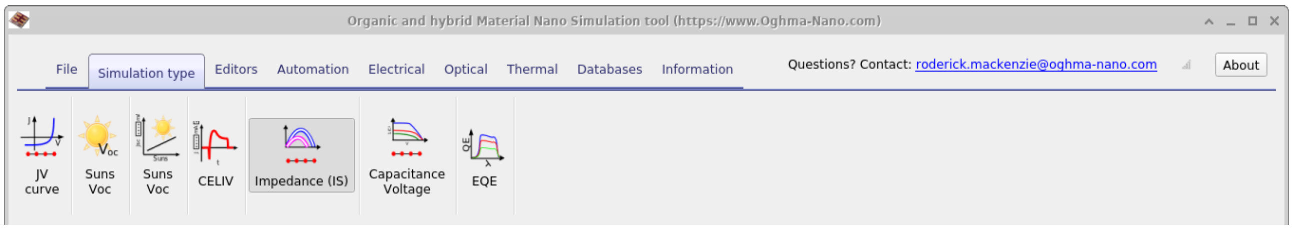


Figure 11.9: Changing the simulation mode to impedance spectroscopy.

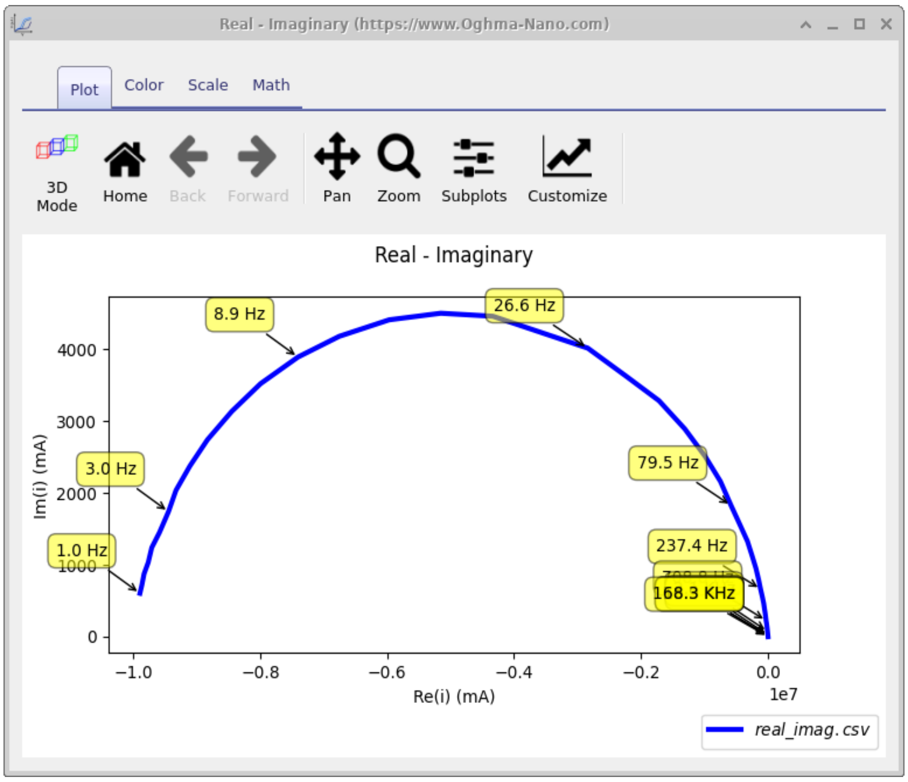


Figure 11.10: An impedance spectroscopy simulation performed using the above circuit. To do this just change the simulation mode to IS.

11.0.2 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 12

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.

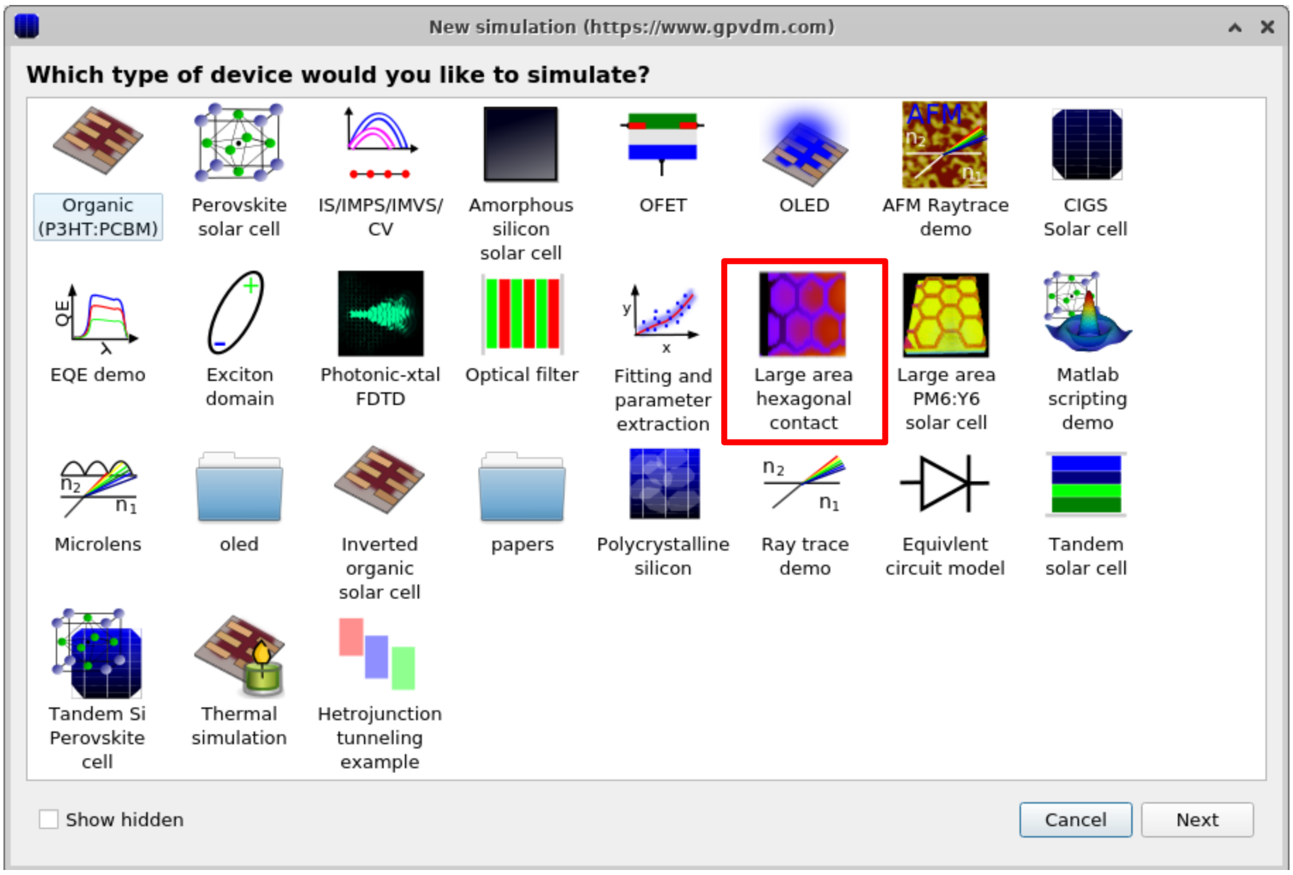
Related YouTube videos:

Tutorial on designing large area contacts for flexible electronics

Understanding Printed Hexagonal Contacts for Large Area Solar Cells

12.1 Designing contacts for large area devices

A common problem is designing large area contacts for solar cells. This paper [21] 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 12.1). Once you have opened it you should get a window which looks like figure 12.2. 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.


New simulation (https://www.gpvdm.com)

Which type of device would you like to simulate?

Figure 12.1: Selecting the large area contact simulation

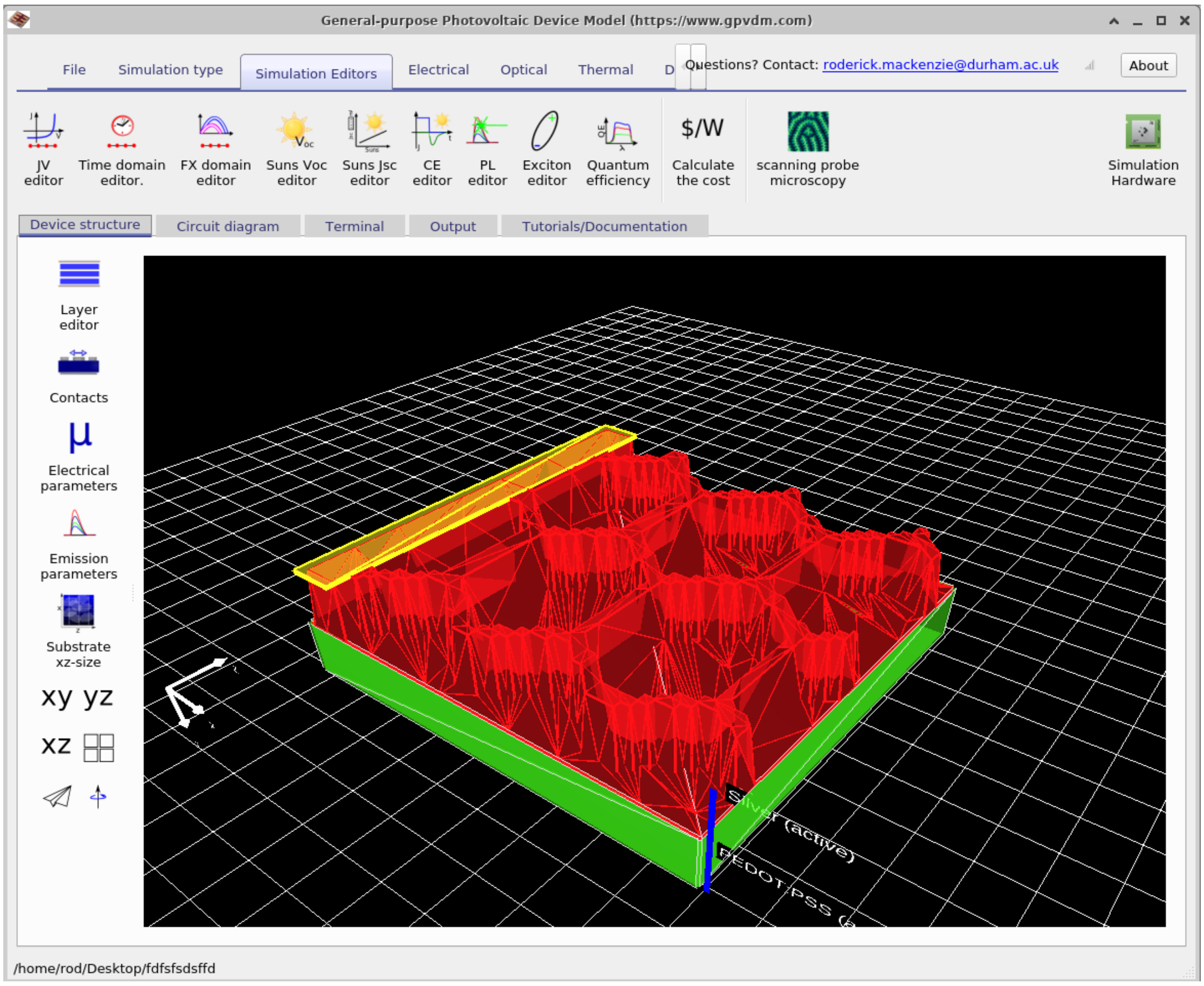


Figure 12.2: A 3D image of the contact printed contact.

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 12.3. Here you can zoom in and examine the individual resistors, each line represents a resistor.

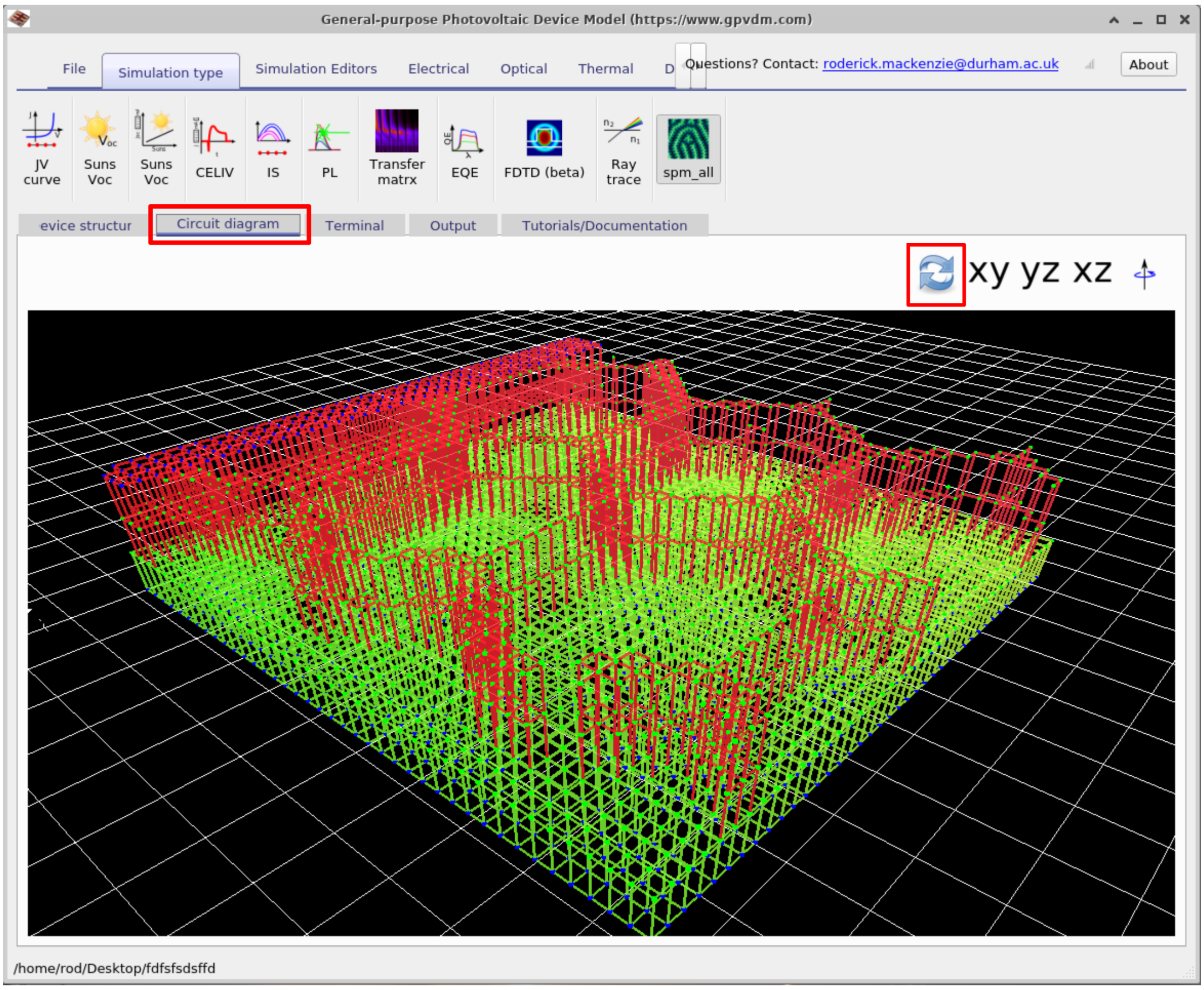
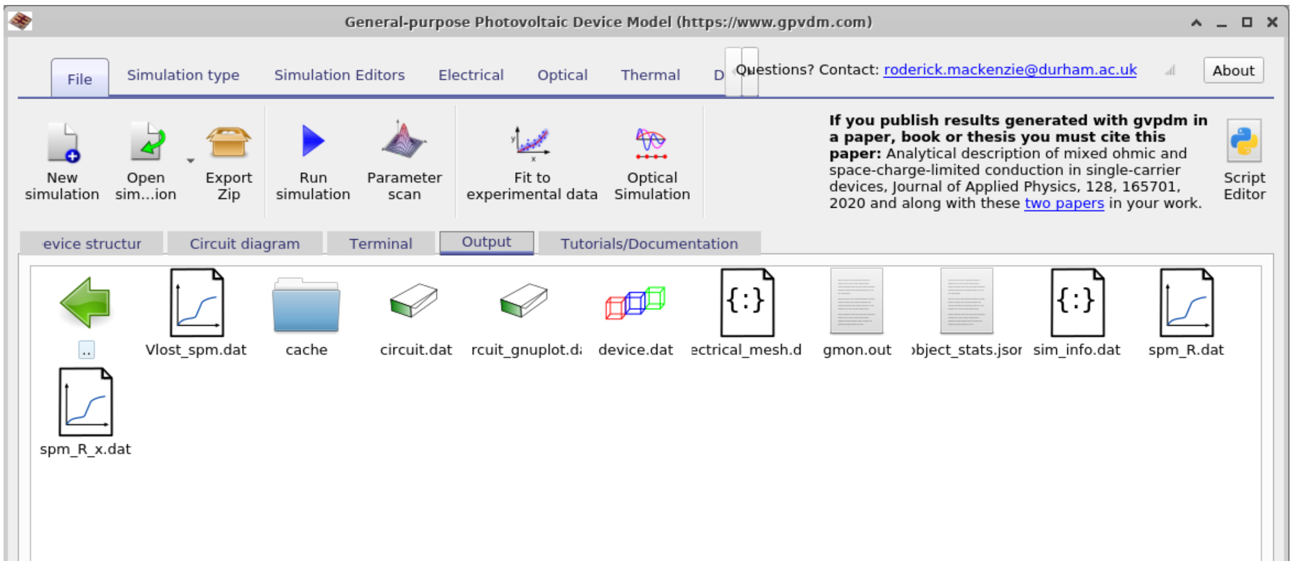


Figure 12.3: Building the 3D circuit mesh of the contact structure.

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 12.2) and the extracting silver contact (far left yellow strip on the top of figure 12.2). Run the simulation by clicking on the play button in the file ribbon.



If you publish results generated with gvpdm in a paper, book or thesis you must cite this paper: Analytical description of mixed ohmic and space-charge-limited conduction in single-carrier devices, Journal of Applied Physics, 128, 165701, 2020 and along with these two papers in your work.

Figure 12.4: 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 12.5. Other output files are listed below in table 12.1.

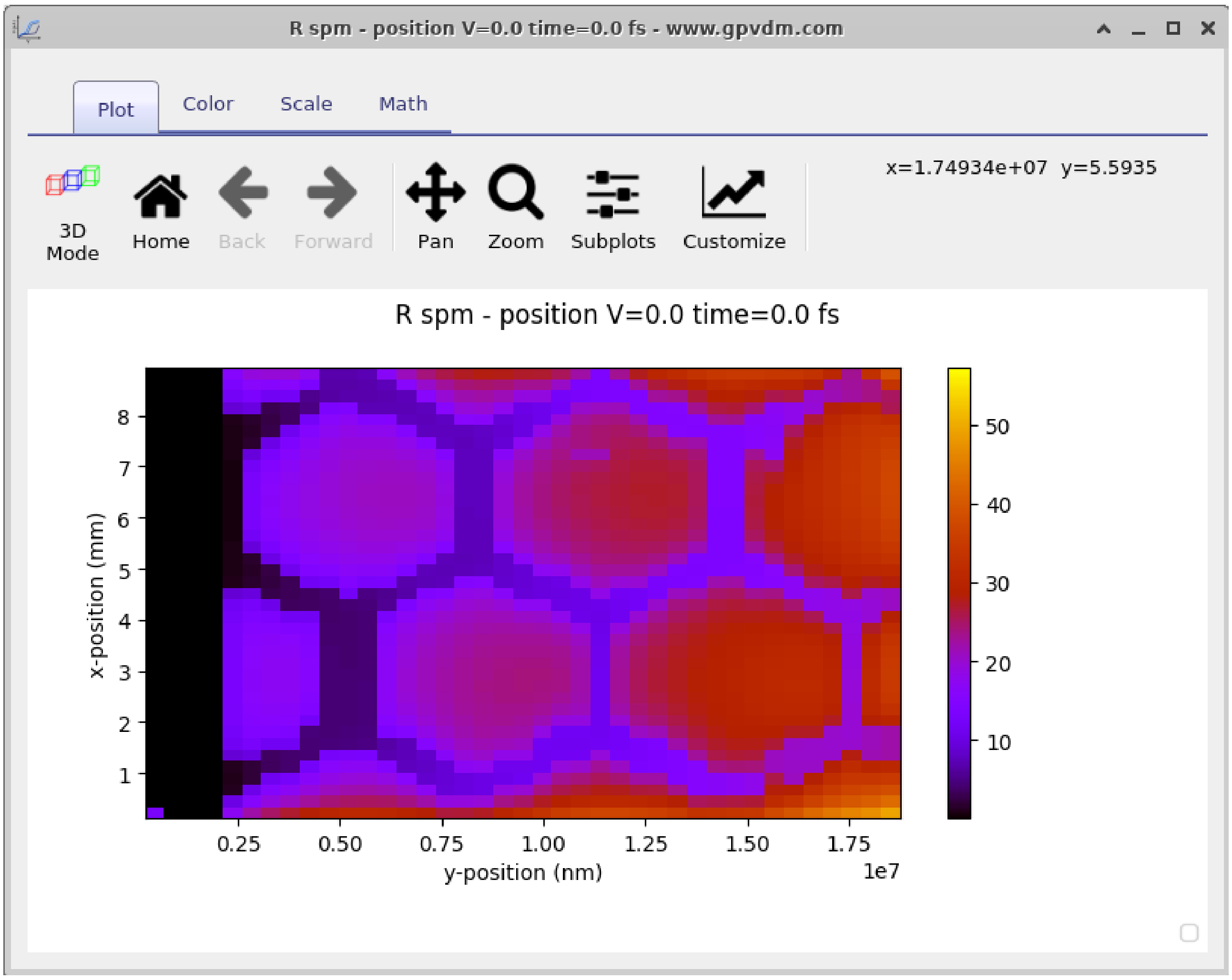


Figure 12.5: A 2D resistance plot across the surface of the device.

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

Table 12.1: Files produced by the SPM simulaton.

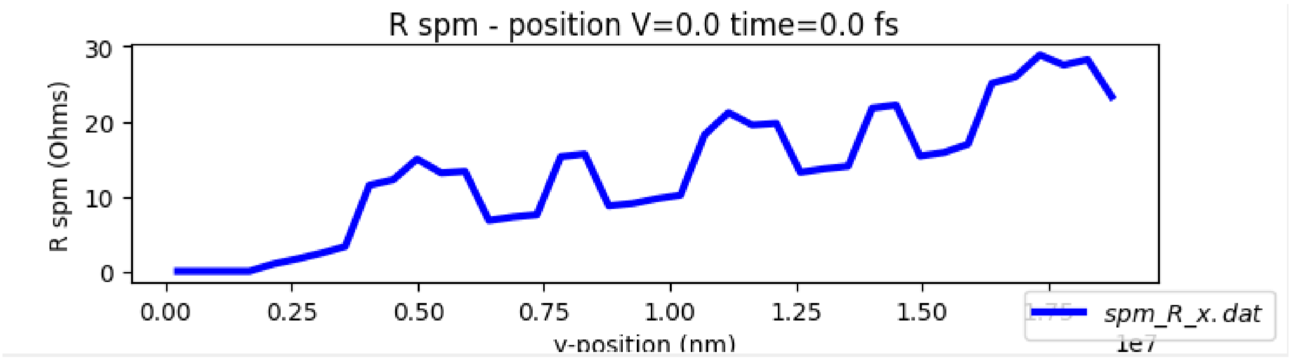
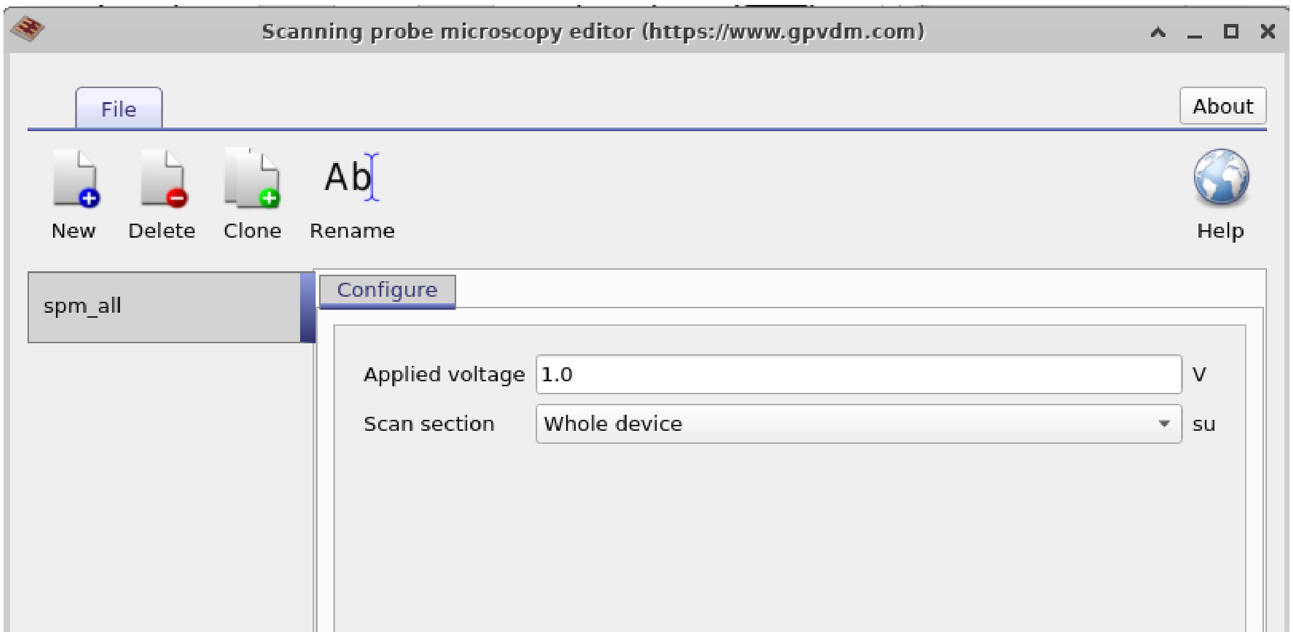


Figure 12.6: A 1D resistance plot taken through the centre of the device.

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 12.7



Applied voltage 1.0 V
Scan section Whole device su

Figure 12.7: The output files from the simulation.

12.2 Simulating large area solar cells

Chapter 13

Modelling excitons/geminate recombination - organics only

13.1 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.

$$G = G_{abs} \cdot \eta_{photon} \tag{13.1}$$

where G is the charge carrier generation rate in $m^{-3}s^{-1}$ in equations 13.1 and 9.26.

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..

13.2 Modelling excitons

So if you have read section 13.1 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 13.1. If the exciton model is turned on then optical absorption will feed straight into the exciton diffusion equation.

$$\frac{\partial X}{\partial t} = \nabla \cdot D \nabla X + G_{optical} - k_{dis}X - k_{FRET}X - k_{PL}X - \alpha X^2. \tag{13.2}$$

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}

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 Föster 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

$$D = \frac{L^2}{\tau} \tag{13.3}$$

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

13.3 Modeling excitions in a device

13.4 Modeling excitions in a unit cell

Chapter 14

The oghma file format

14.1 the .oghma simulation file format

In OghamNano 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 14.2. Also have a look at the file in notepad to get a sense of what is in it.

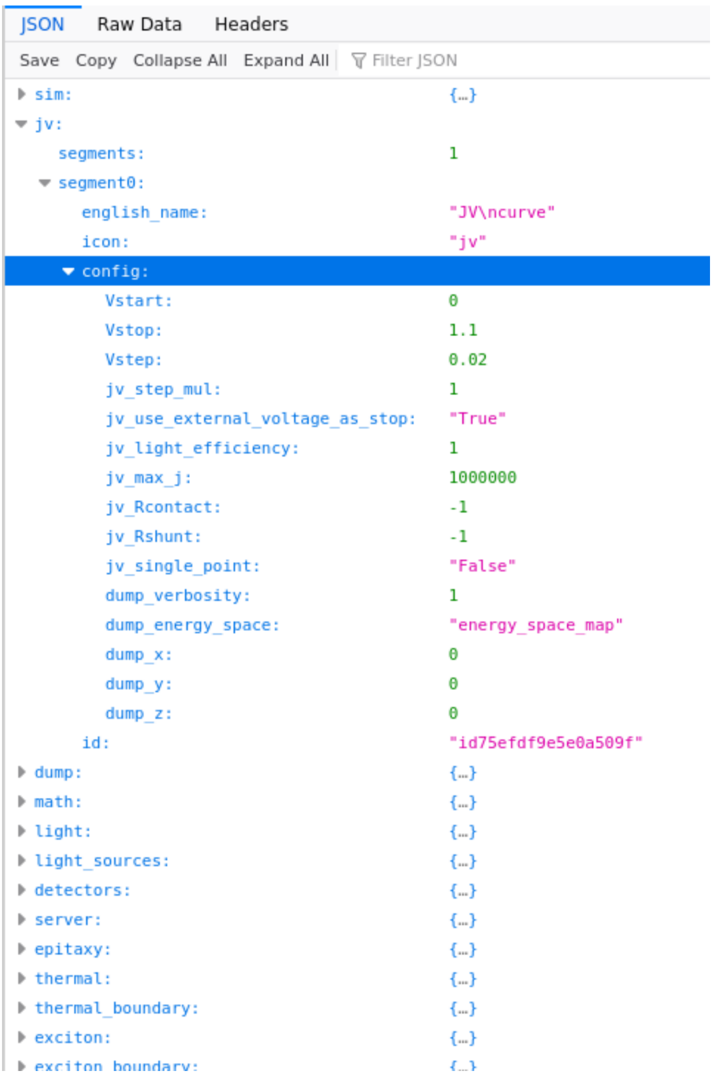
```
{
  "color_of_dog": "brown",
  "dog_age": 5,
  "dogs_toys": {
                "rabbit": "True",
                "stick": "False"
                }

}
```

Figure 14.1: A simple JSON example

You can see that the json file has various headings, key headings are listed below in table 14.1. If you wish to programmatically drive OghmaNano you can simply use one of the many available json editors most languages have them freely available.

14.2 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



```
▸ sim:                                  {…}
▾ jv:
    segments:                           1
  ▾ segment0:
      english_name:                     "JV\ncurve"
      icon:                             "jv"
    ▾ config:
        Vstart:                         0
        Vstop:                          1.1
        Vstep:                          0.02
        jv_step_mul:                    1
        jv_use_external_voltage_as_stop: "True"
        jv_light_efficiency:            1
        jv_max_j:                       1000000
        jv_Rcontact:                    -1
        jv_Rshunt:                      -1
        jv_single_point:                "False"
        dump_verbosity:                 1
        dump_energy_space:              "energy_space_map"
        dump_x:                         0
        dump_y:                         0
        dump_z:                         0
      id:                               "id75efdf9e5e0a509f"
▸ dump:                                 {…}
▸ math:                                 {…}
▸ light:                                {…}
▸ light_sources:                        {…}
▸ detectors:                            {…}
▸ server:                               {…}
▸ epitaxy:                              {…}
▸ thermal:                              {…}
▸ thermal_boundary:                     {…}
▸ exciton:                              {…}
▸ exciton_boundary:                     {…}
```

Figure 14.2: 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 14.1: Key headings/sections in the sim.json file.

seen in figure 14.2 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 14.2. 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.

14.3 Encoding

The .json files read/written by OghamNano 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.

14.4 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 15

Databases

There are a series of databases used to define material parameters, shapes 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 15.1.

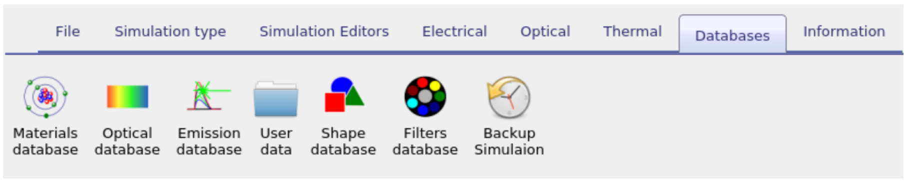


Figure 15.1: The database ribbon

There are two copies of these databases, one copy in the install directory of OghmaNano C:\Program Files\OghmaNano\ and one in your home directory in a folder called oghma_local. When the model starts for the first time it copies the read only materials database from, to the oghma_local folder in your home directory. If you delete the copy of the materials database in the oghma_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 oghma_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. Within each directory there is a data.json file which defines basic material properties of the material such as what it is and what icon to use for it. There may be a couple of .bib files that contain reference information for the object in bibtex format. The rest of the key information will be stored in human readable .csv files. These files can be opend in notepad or any text editor. The one exception is that in the shape database some large files are stored in a binary format.

15.1 Materials database

Related YouTube videos:

A tutorial on adding new materials to OghmaNano

This database primarily contains n/k data and PL emission spectra. However it also contains some electrical information and some thermal information. Each subdirectory within the materials database identifies the material name. In each sub directory there are two key files *alpha.csv* and *n.csv*, these files are standard text files can be opened with any text editor such as wordpad. Alpha.csv contains the absorption coefficient of the material while n.csv 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.csv) and the real part of the refractive index (i.e. n) in au (for n.csv). The data.json defines the material color and any known electrical or thermal data. If the material is used in emissive optical simulations the emission spectra of the material will be stored in a file called *emission.csv*. An example material directory, in this case Alq3 can be seen in figure 15.2, a description of these files can be found in 15.1.

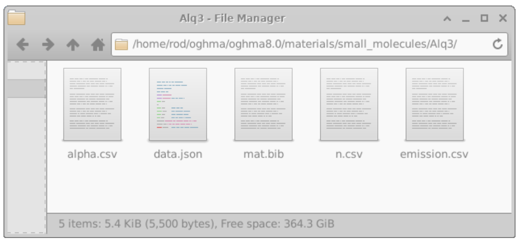

Alq3 - File Manager
/home/rod/oghma/oghma8.0/materials/small_molecules/Alq3/
alpha.csv
data.json
mat.bib
n.csv
emission.csv
5 items: 5.4 KiB (5,500 bytes), Free space: 364.3 GiB

Figure 15.2: An example of a materials database item for Alq3.

File name	Description
data.json	json file infomration about the material (LUMO/HOMO levels etc..)
alpha.csv	wavelength (m) v.s. absorption (m^{-1})
n.csv	wavelength (m) v.s. refractive index (au)
$emission.csv^{*seebelow}$	wavelength (m) v.s. PL emission (au)
mat.bib	Bibtex file containing references

Table 15.1: A summary of the files making up each material in the materials database. *The file emssion.csv is not needed unless the material forms part of an emissive layer of an OLED or other such light emitting device.

15.1.1 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 oghma_local\materials\oxieds to a new directory (say oghma_local\materials\oxieds\mynewmaterial). Then replace alpha.csv and n.csv 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.

15.1.2 Adding new materials - the easy way

To add a new material go to the database ribbon and click on *Materials database* as shown in figure 15.3.

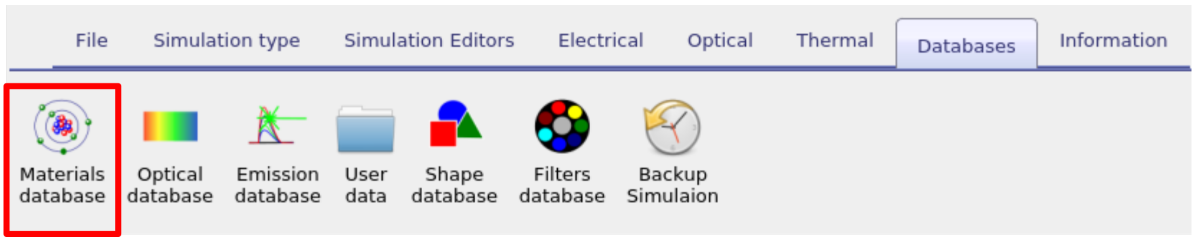
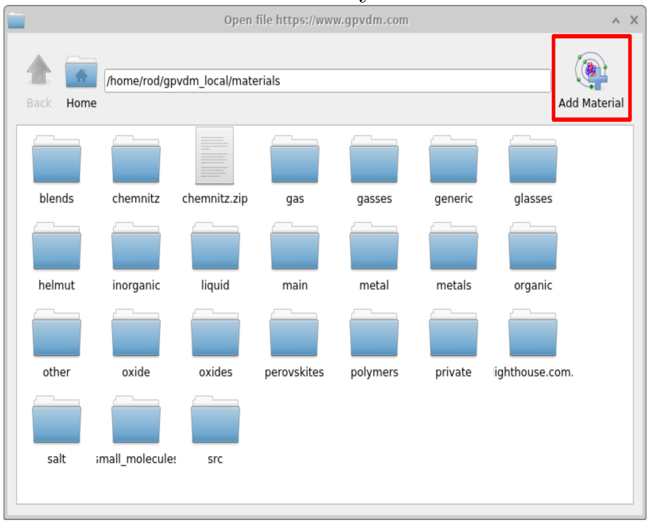


Figure 15.3: Opening the materials database

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 15.5. In this case we called the material my_new_material.



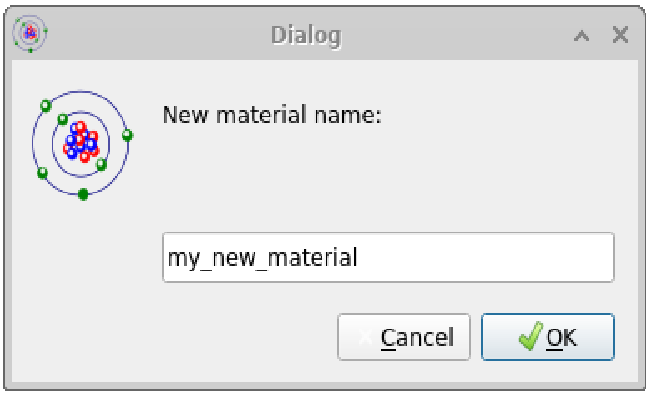
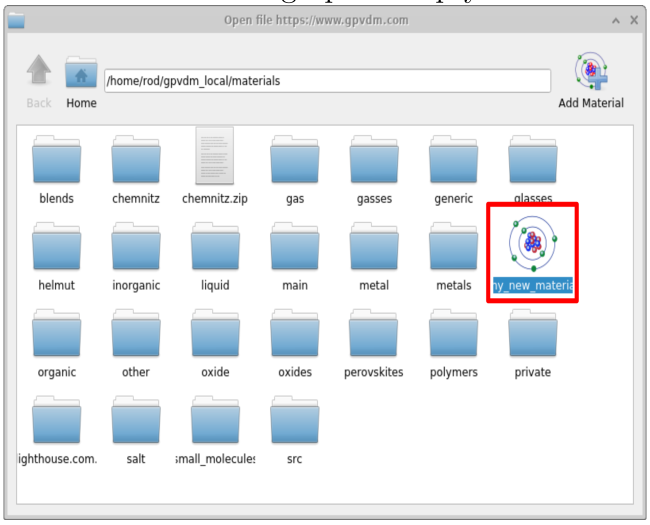


Figure 15.5: Type the name of the new material

Figure 15.4: Select Add material

Once you have clicked OK the new material will appear see Figure 15.6, open it by double clicking on it. This will bring up an empty material window with no data. See Figure 15.7.



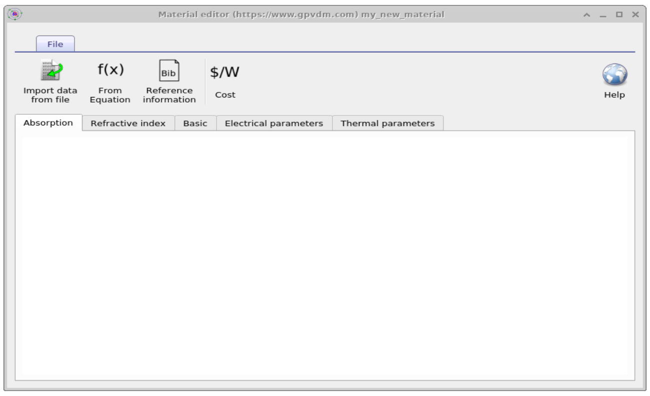
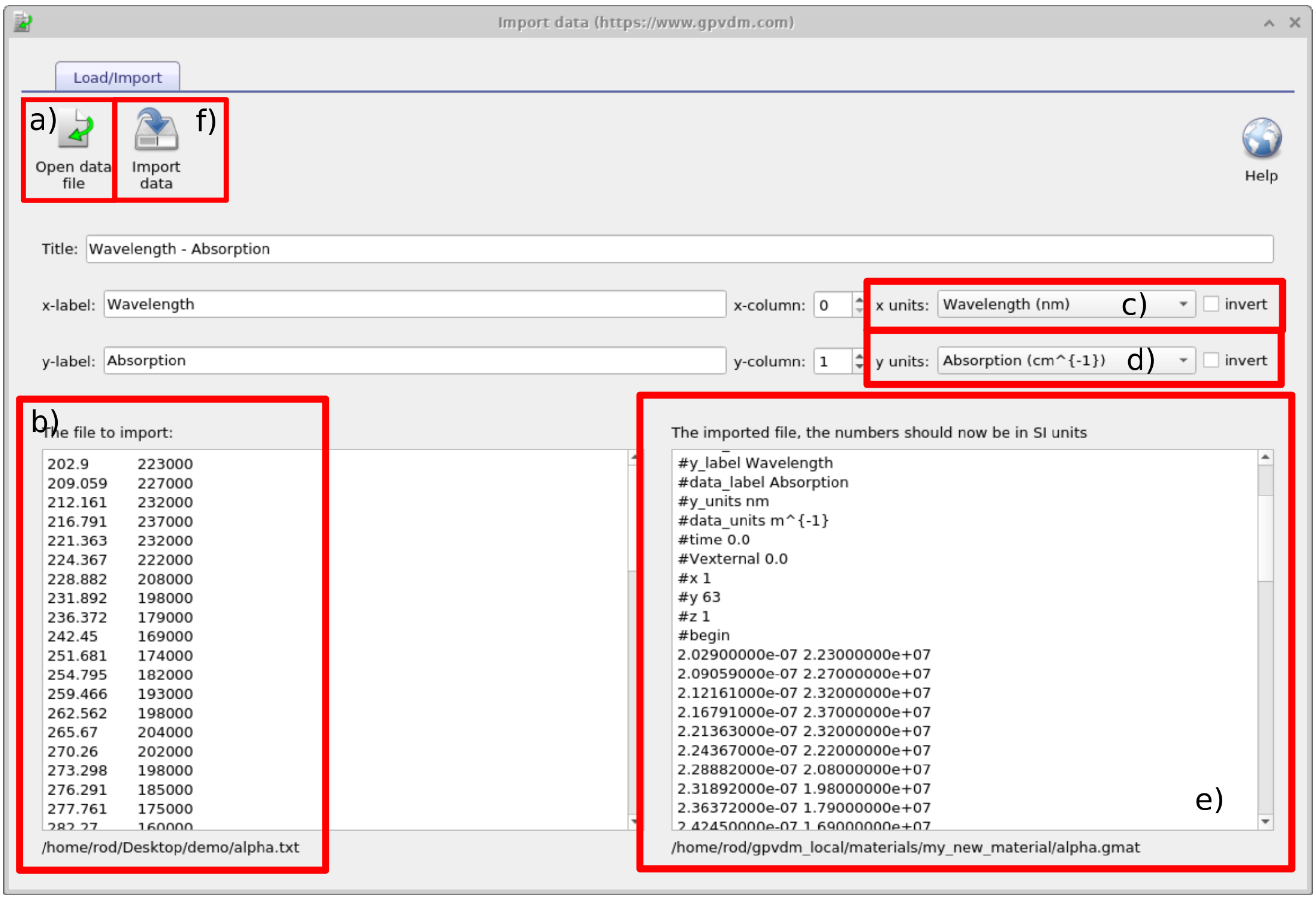


Figure 15.7: The new material without any data

Figure 15.6: Open the new material

To import data to the material use the *Import data from file* button situated on the top left of the material window see Figure 15.7 this will bring up the import data wizard which is shown in Figure 15.8. To use this wizard follow these steps:



Title: Wavelength - Absorption

x-label: Wavelength x-column: 0 x units: Wavelength (nm) invert

y-label: Absorption y-column: 1 y units: Absorption (cm^{-1}) invert

The file to import:

```
202.9      223000
209.059    227000
212.161    232000
216.791    237000
221.363    232000
224.367    222000
228.882    208000
231.892    198000
236.372    179000
242.45     169000
251.681    174000
254.795    182000
259.466    193000
262.562    198000
265.67     204000
270.26     202000
273.298    198000
276.291    185000
277.761    175000
282.27     160000
```

/home/rod/Desktop/demo/alpha.txt

The imported file, the numbers should now be in SI units

```
#y_label Wavelength
#data_label Absorption
#y_units nm
#data_units m^{-1}
#time 0.0
#Vexternal 0.0
#x 1
#y 63
#z 1
#begin
2.02900000e-07 2.23000000e+07
2.09059000e-07 2.27000000e+07
2.12161000e-07 2.32000000e+07
2.16791000e-07 2.37000000e+07
2.21363000e-07 2.32000000e+07
2.24367000e-07 2.22000000e+07
2.28882000e-07 2.08000000e+07
2.31892000e-07 1.98000000e+07
2.36372000e-07 1.79000000e+07
2.42450000e-07 1.69000000e+07
```

/home/rod/gpvdm_local/materials/my_new_material/alpha.gmat

Figure 15.8: The data importer window

a) Open a file you want to import. The file can only be a text file or a csv file.

b) Once the file has been loaded it will be visible in the text box on the left.

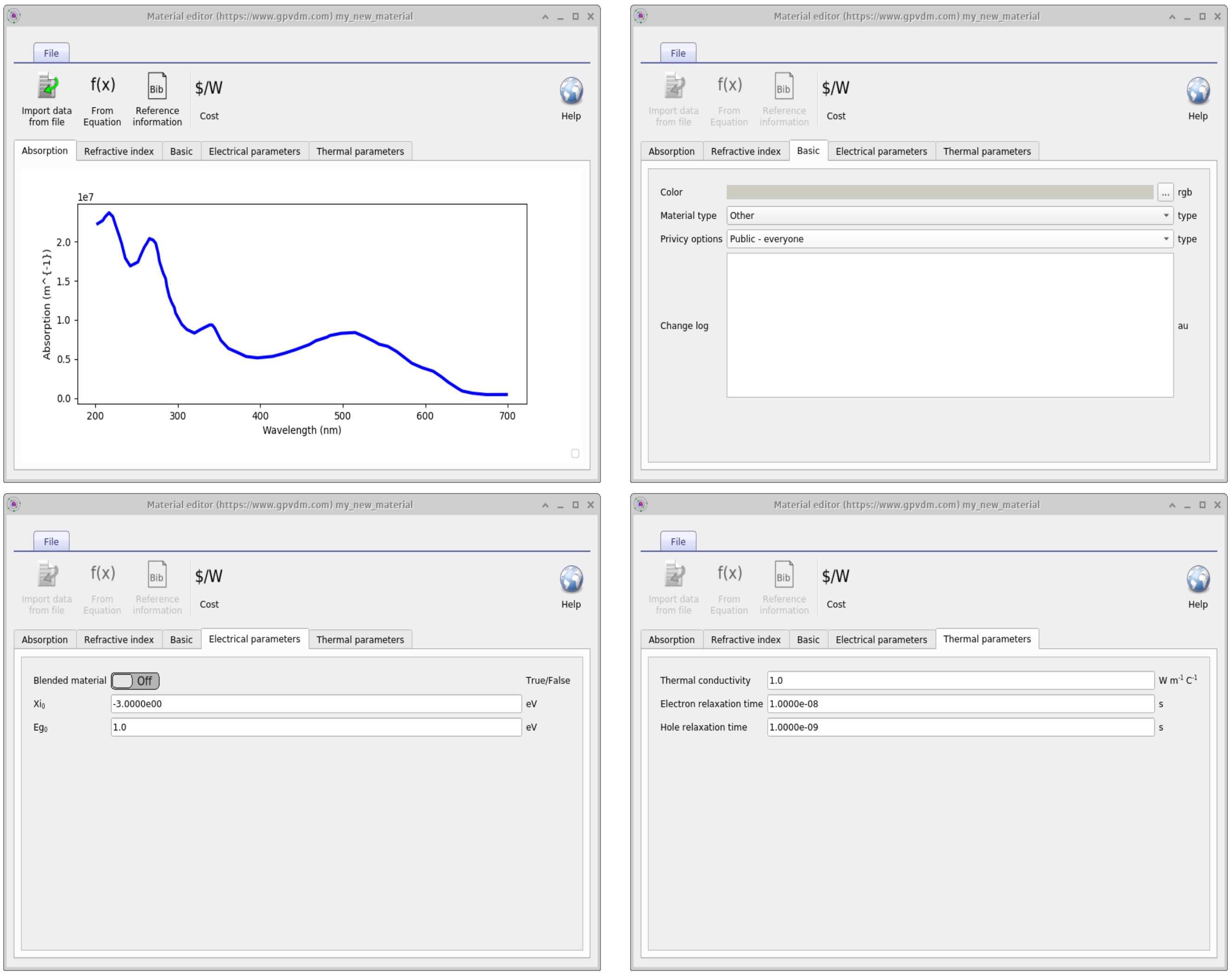
c) Select the units of the x-axis of the original file.

d) Select the units of the y-axis of the original file.

e) The file should appear converted into SI units on the right hand text box.

f) If you have happy with the conversion click import data and the data will be saved.

This process can be seen in 15.8, once done the imported data will appear in the material as shown on the top left of 15.9. In this example absorption data was imported, for the material to be used in a simulation the refractive index (real part) will also need to be imported.



Color	...	rgb
Material type	Other	type
Privicy options	Public - everyone	type
Change log		au

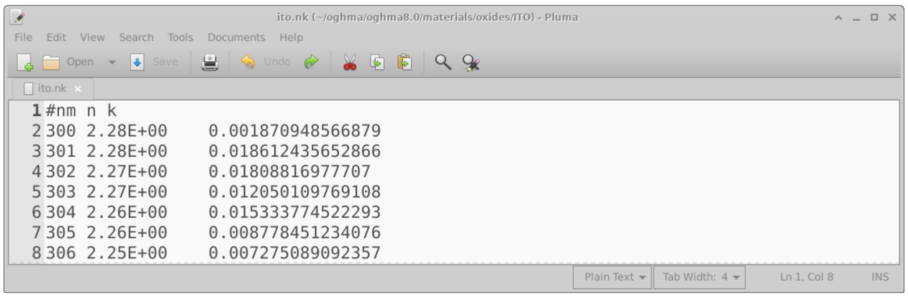
Blended material	Off	True/False
Xi_0	-3.0000e00	eV
Eg_0	1.0	eV

Thermal conductivity	1.0	W m^{-1} C^{-1}
Electron relaxation time	1.0000e-08	s
Hole relaxation time	1.0000e-09	s

Figure 15.9: Clockwise from the top left; The imported absorption spectra; The basic material parameters; The electrical parameters; and the Thermal parameters.

15.1.3 But I have a data in nm/n/k format

Usually OghamNano only accepts data input in SI units and usually only accepts input in one format. This is to reduce the overall number of lines of code and reduce maintenance. However, I have been asked to make the model accept data of format: wavelength (nm), refractive index (au), k (au), as shown in Figure 15.10. OghmaNano will be able to read this file if you simply copy it into your_home_directory\oghma_local\materials and give it a file extension .nk, so for example ito.nk. Where your_home_directory is simply your Windows home directory. It is usually located on the C:\Users on a home PC but the location can change if you are on a corporate PC. Once you have dragged the file into your_home_directory\oghma_local\materials it will appear as a material in the model. See Figure 15.11. You can see that the .nk file in Figure 15.11 is greyed out, this is because it is a non standard OghamNano file, that OghmaNano can read but not edit.



```
#nm n k
300 2.28E+00    0.001870948566879
301 2.28E+00    0.018612435652866
302 2.27E+00    0.01808816977707
303 2.27E+00    0.012050109769108
304 2.26E+00    0.015333774522293
305 2.26E+00    0.008778451234076
306 2.25E+00    0.007275089092357
```

Figure 15.10: An example of wavelength/n/k data.

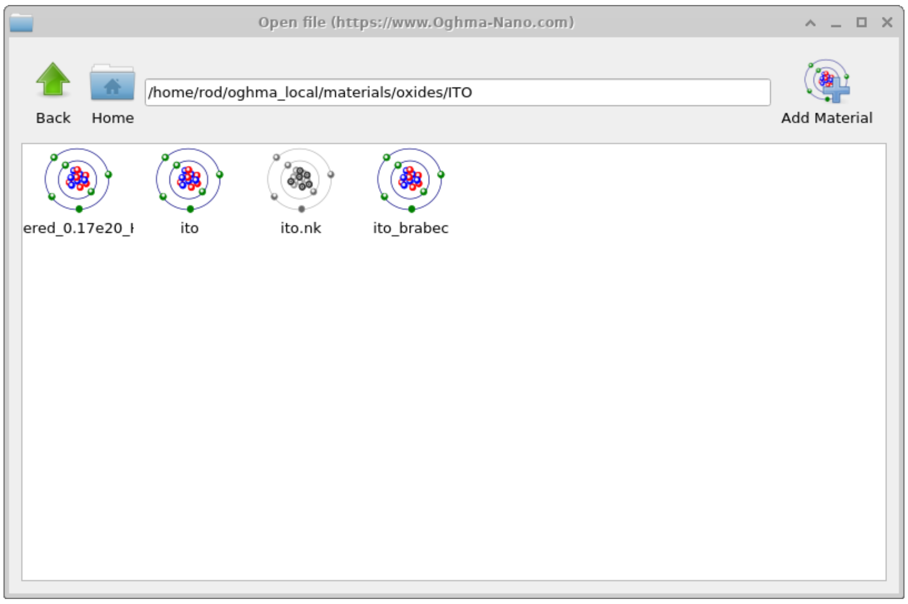

Open file (https://www.Oghma-Nano.com)
Back
Home
/home/rod/oghma_local/materials/oxides/ITO
Add Material
ered_0.17e20_l
ito
ito.nk
ito_brabec

Figure 15.11: An example of a .nk file containing wavelength/n/k data in the materials database. You can see it is greyed out denoting that the file is not in native OghmaNano format.

15.2 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 15.12.

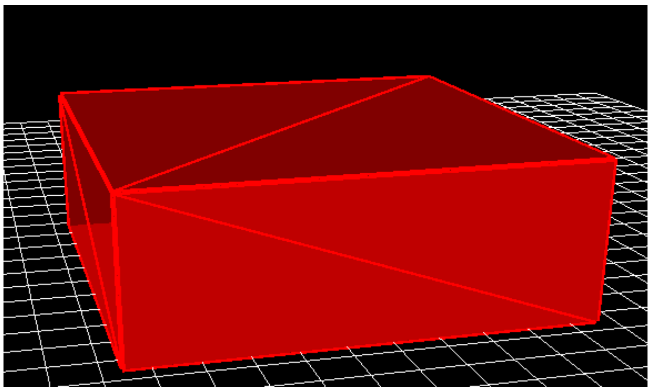


Figure 15.12: the box *shape*.

Shapes are stored in the shape database, this can be accessed via the database ribbon and clicking on the Shapes icon, see figure 15.13. By clicking on the *shape database* icon the shape database window can be brought up see figure 15.14.

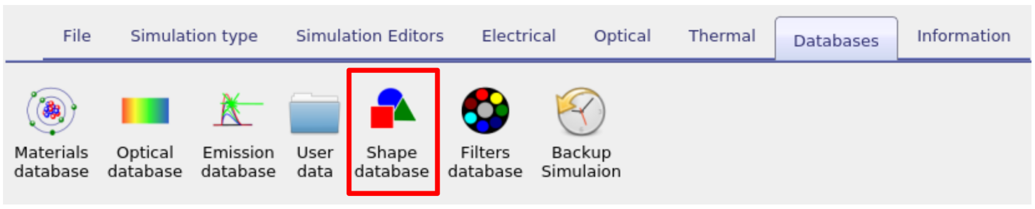


Figure 15.13: Opening the shape database

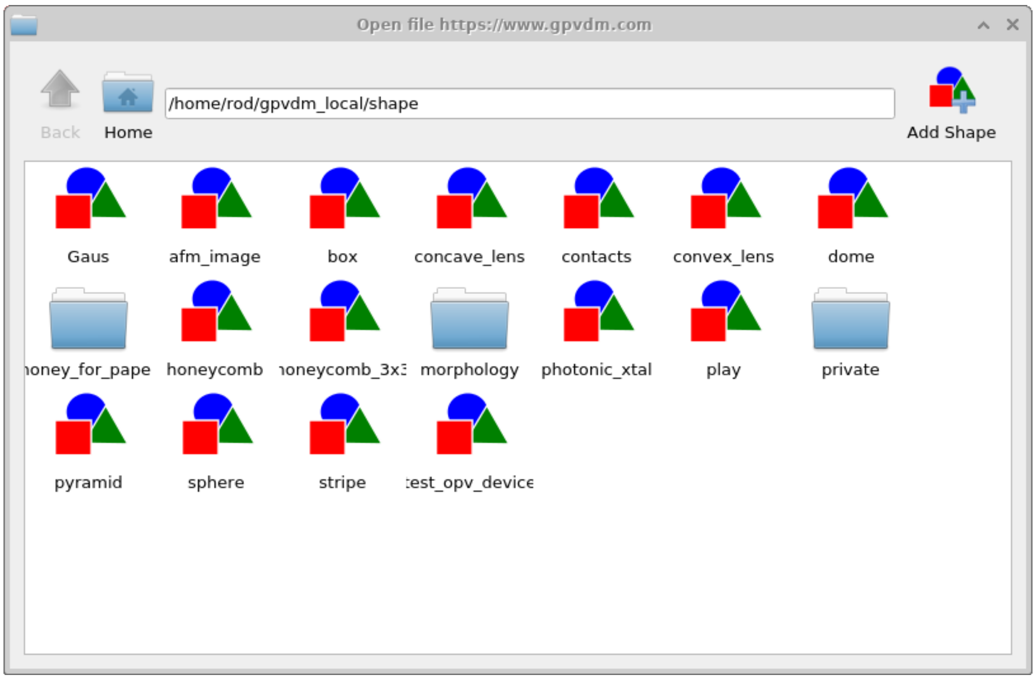


Figure 15.14: The shape database window

Try opening some of the shapes and have a look at them. You will get a window much like that shown in figure 15.15. Figure 15.15 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.

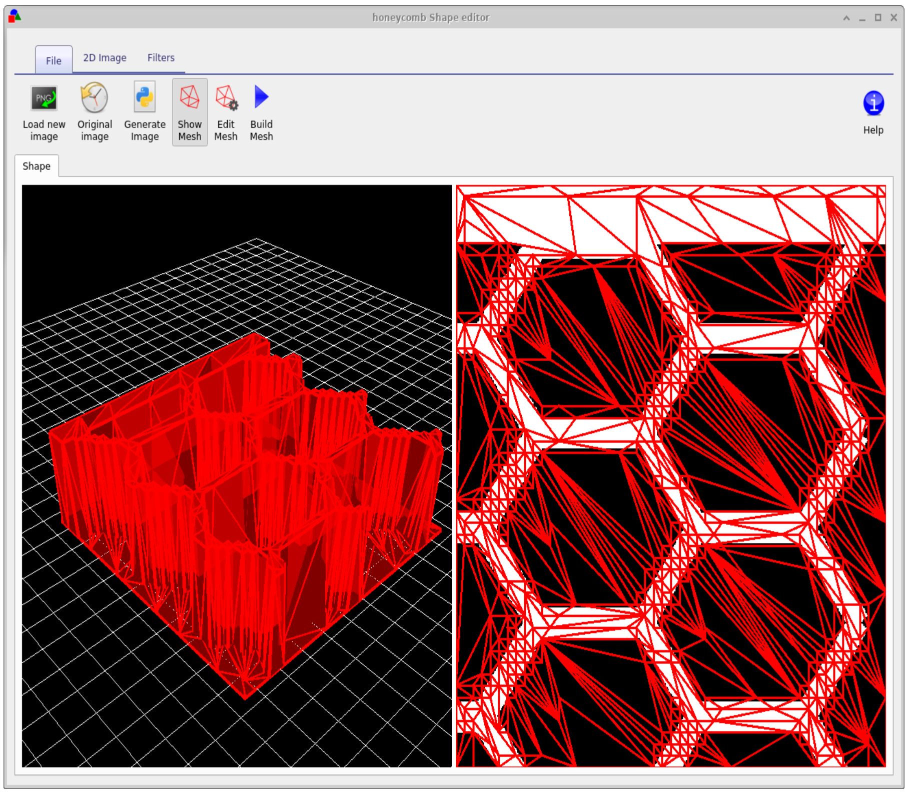


Figure 15.15: An example of a shape generated from a 2D png image. The 3D shape representing a hexagonal contact from a solar cell is on the left of the figure while the original 2D image is on the right.

Now try opening the shape *morphology/1* and you should see a window such as the one shown in figure 15.16, 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 15.17. 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.

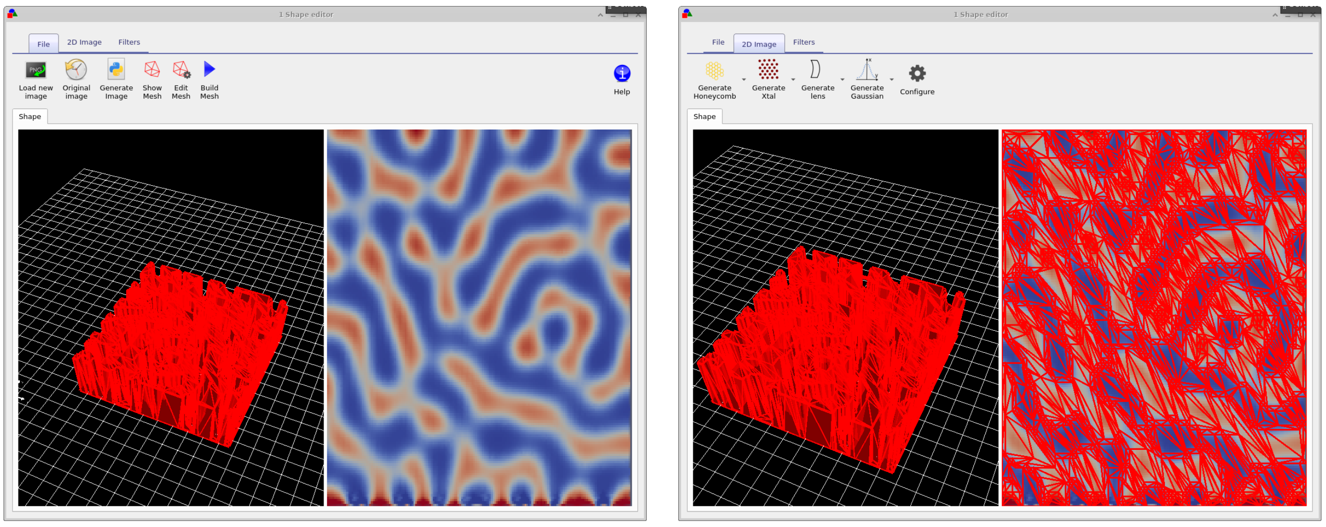
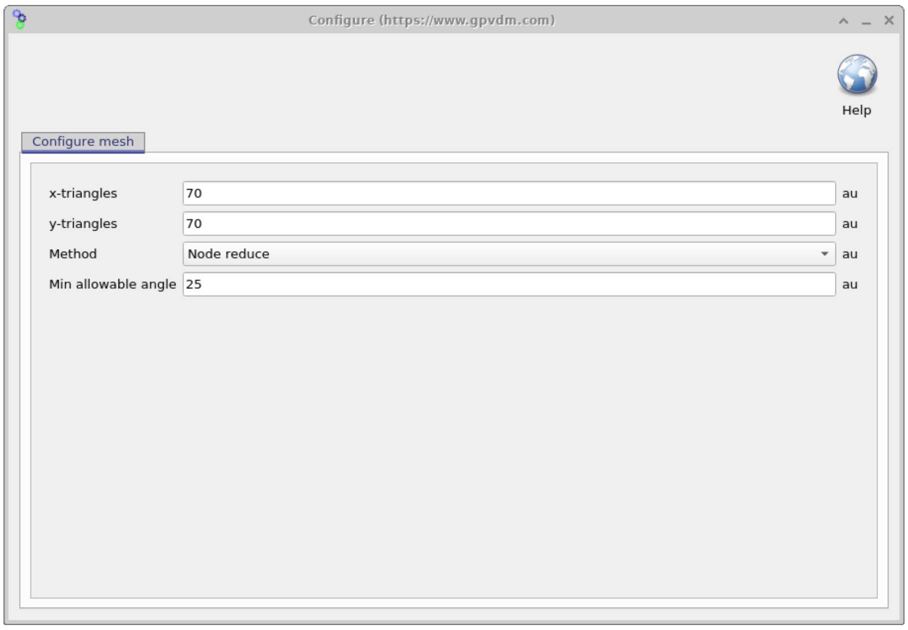


Figure 15.16: Clockwise from the top left; The imported absorption sepctrum; The basic material parameters; The electrical parameters; and the Thermal parameters.

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.



x-triangles	70	au
y-triangles	70	au
Method	Node reduce	au
Min allowable angle	25	au

Figure 15.17: 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 15.16. 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.

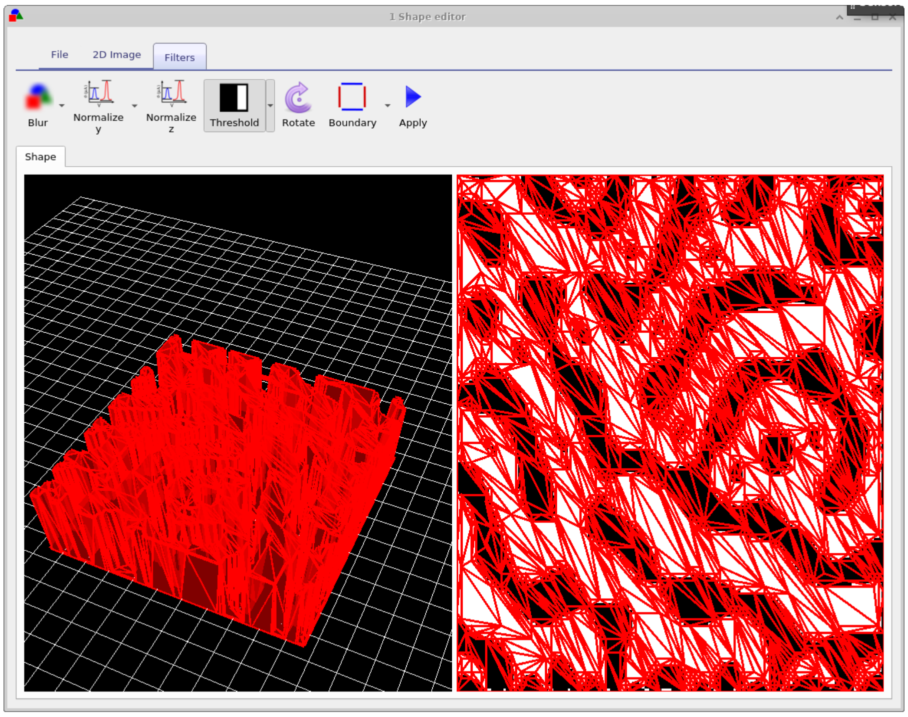


Figure 15.18: The shape database

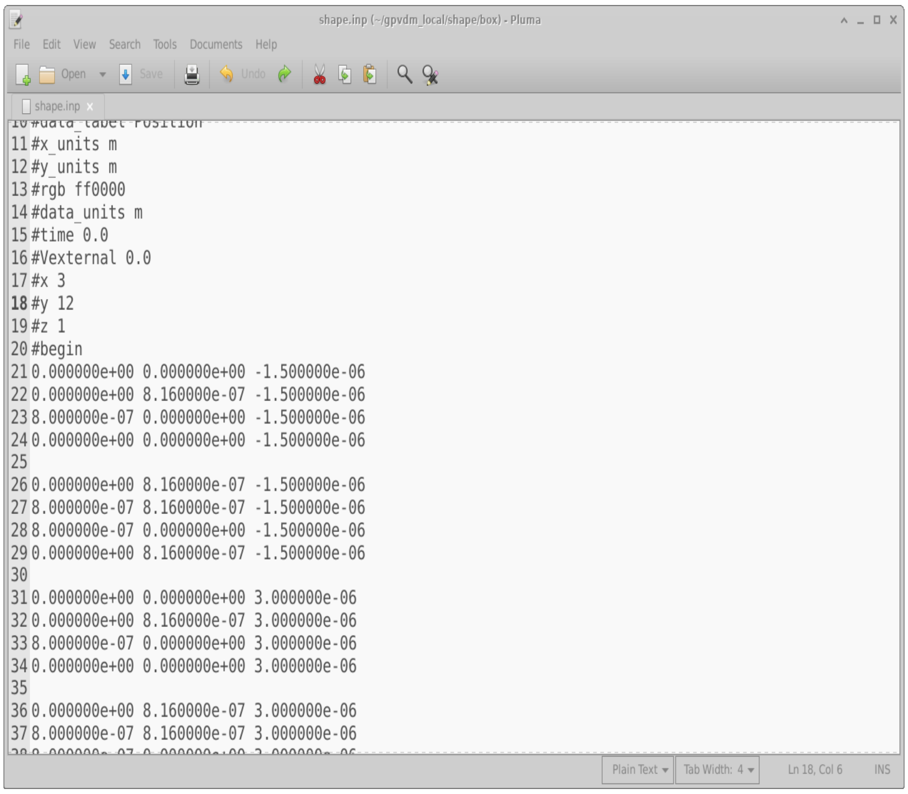
15.2.1 The shape file format

A shape has to be a fully enclosed volume, if you use the built in shape discretizer this will be done for you automaticity. 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
data.json	Holds the configuration for the shape file
image_original.png	backup of the imported image
image_out.png	The final processed image
image.png	The imported image which may be modified.
shape.inp	The discretized 3D structure.

Table 15.2: 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 15.19. 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 forth line is a repeat of the first line so that gnuplot can plot the triangle nicely. The number 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
Plain Text Tab Width: 4 Ln 18, Col 6 INS

```
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 8.000000e-07 0.000000e+00 3.000000e-06
```

Figure 15.19: An example of the shape.inp file.

15.3 Filters database

This contains optical filters.

15.4 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 15.1), this will bring up the backup window, see figure 15.20. 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..

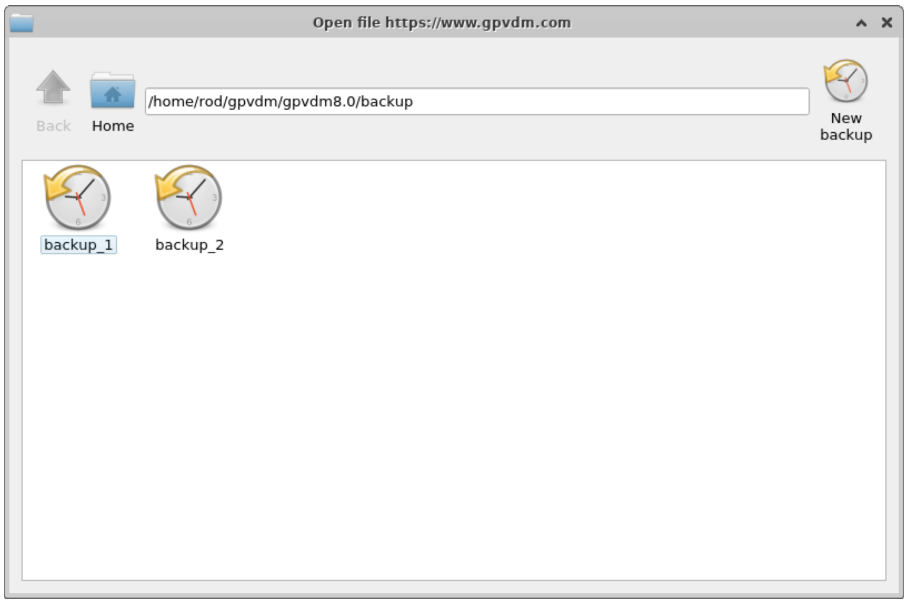

Open file https://www.gpvdm.com
Back
Home
/home/rod/gpvdm/gpvdm8.0/backup
New backup
backup_1
backup_2

Figure 15.20: Backing up a simulation

Chapter 16

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 OghmaNan

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 fit to experimental data using the fitting function. Fitting is a good way to extract physical parameters from a device such as mobility, destiny 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 OghamNano.

16.1 Key tips and tricks

- Generally speaking fitting is 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 and order of magnitude go back and think again about what you are asking the model to do. For example if you just can't get J_{sc} to match on a solar cell, could it be that your material just just not absorbing enough photons to get your desired J_{sc} value?
- 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/tale 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}. Alway 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

; 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.

16.2 The main fitting window

An example of how to fit the model to experimental data is included in the demo simulations provided with OghmaNano. In this example a simple drift diffusion model is fit to some experimental data. If you click on the *New simulation* icon in the File ribbon, this will bring the new simulation window as seen in Figure 16.1a. Double click on the *Scripting and fitting* icon, this will display the menu that can be seen in Figure 16.1b. From this menu double click on the *Fitting and parameter extraction* example.(see 16.1a). If you open this simulation you will be presented with the window shown in Figure (see 16.2), this is a simple solar cell simulation. It should be noted that the fitting engine can be used to fit any simulation to any data set.

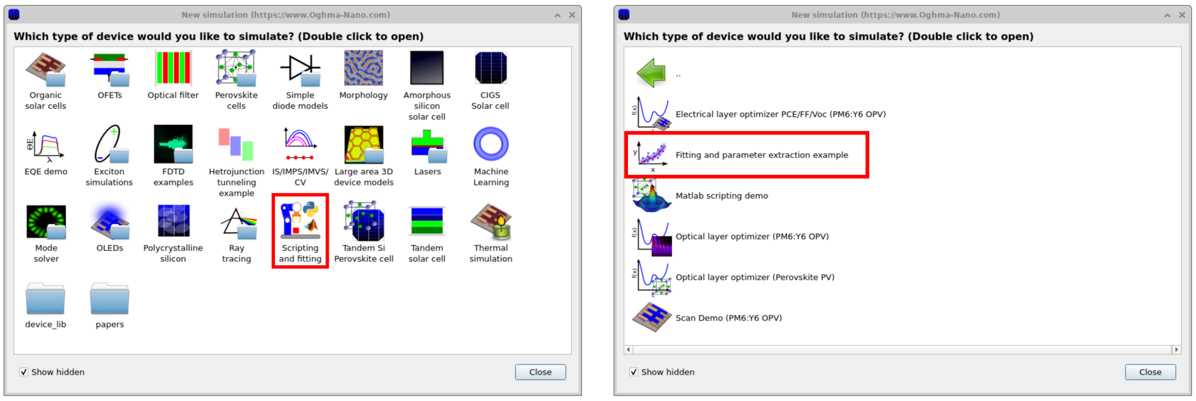
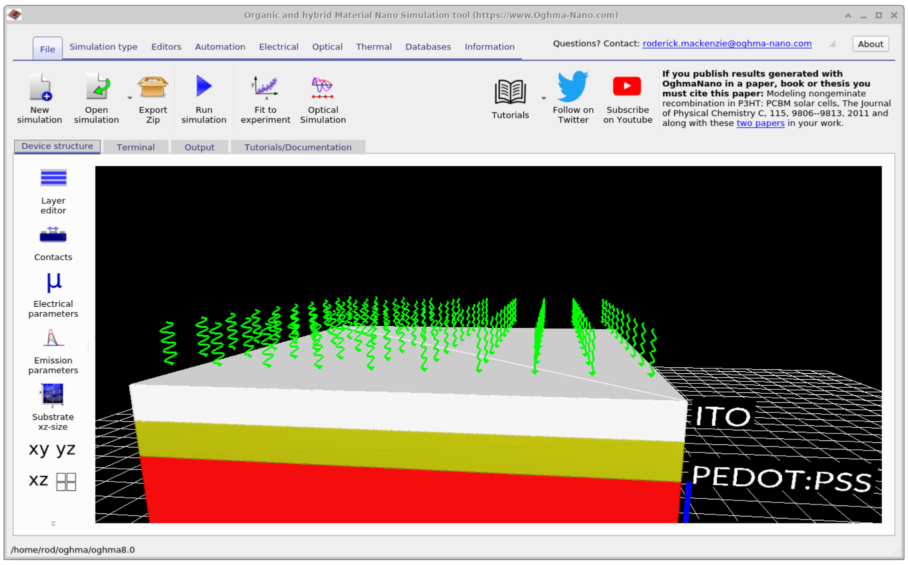

New simulation (https://www.Oghma-Nano.com)
Which type of device would you like to simulate? (Double click to open)
Organic solar cells
OFETs
Optical filter
Perovskite cells
Simple diode models
Morphology
Amorphous silicon solar cell
CIGS Solar cell
EQE demo
Exciton simulations
FDTD examples
Hetrojunction tunneling example
IS/IMPS/IMVS/ CV
Large area 3D device models
Lasers
Machine Learning
Mode solver
OLEDs
Polycrystalline silicon
Ray tracing
Scripting and fitting
Tandem Si Perovskite cell
Tandem solar cell
Thermal simulation
device_lib
papers
Show hidden
Close
New simulation (https://www.Oghma-Nano.com)
Which type of device would you like to simulate? (Double click to open)
..
Electrical layer optimizer PCE/FF/Voc (PM6:Y6 OPV)
Fitting and parameter extraction example
Matlab scripting demo
Optical layer optimizer (PM6:Y6 OPV)
Optical layer optimizer (Perovskite PV)
Scan Demo (PM6:Y6 OPV)
Show hidden
Close

Figure 16.1: a) The example fitting simulation can be found in the *Scripting and fitting* folder b) The fitting example is called *Fitting and parameter extraction example.*



If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: Modeling nongeminate recombination in P3HT: PCBM solar cells, The Journal of Physical Chemistry C, 115, 9806--9813, 2011 and along with these two papers in your work.

Figure 16.2: The main fitting window.

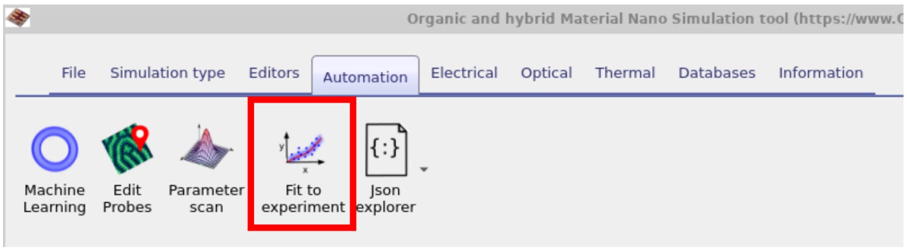
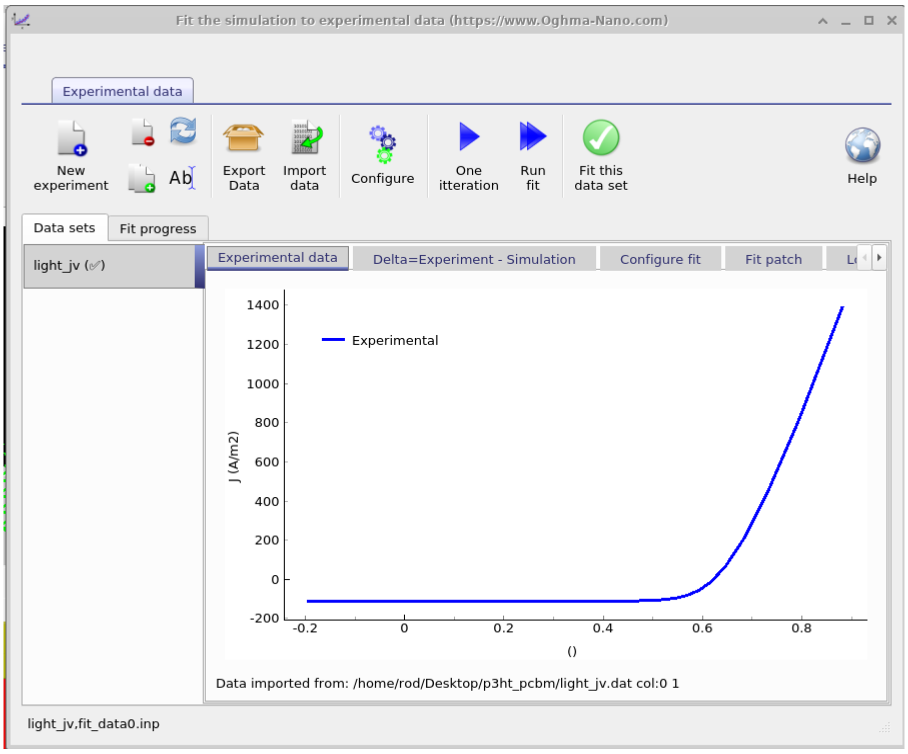


Figure 16.3: The fitting window can be accessed through the *Automation* ribbon, using the *Fit to experiment* icon.

From the *Automation* ribbon in the simulation (Figure 16.3), click on the *Fit to experiment* icon. This will bring up the fitting window (see figure 16.4). The icons in the ribbon of the fitting 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.

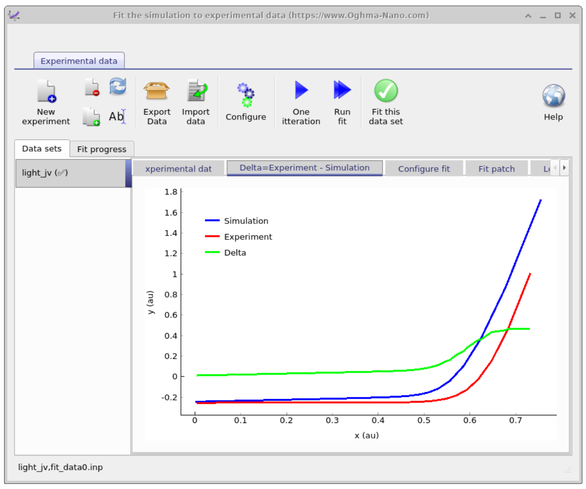


Data imported from: /home/rod/Desktop/p3ht_pcbm/light_jv.dat col:0 1

light_jv,fit_data0.inp

Figure 16.4: The main fitting window.

If you click the one fit button, the fit window will update and will look like 16.5a. 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 16.5b. This process should take about 30 seconds, if it takes longer read section 16.1 above.



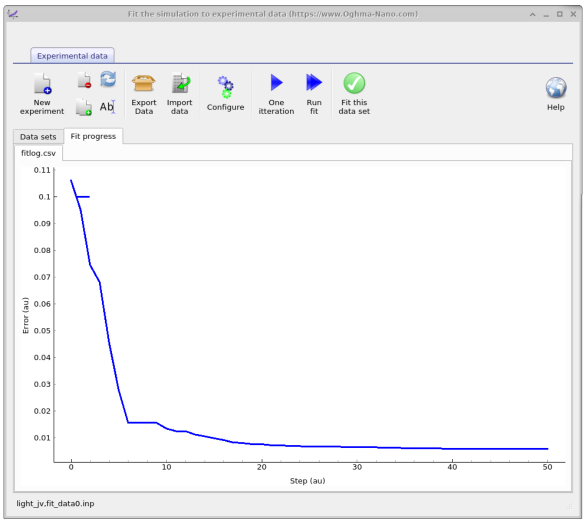
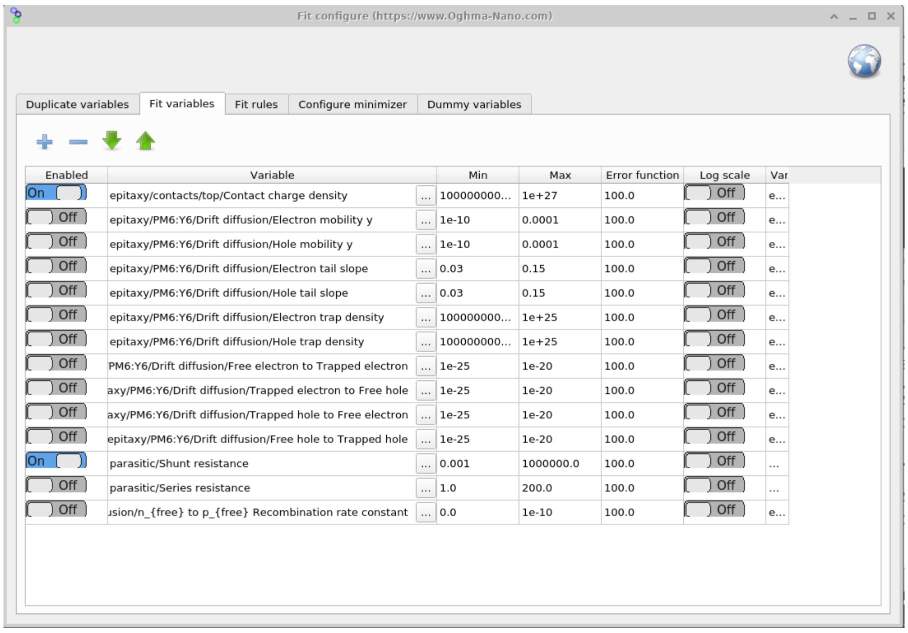


Figure 16.5: a) The result of clicking the one fit button. b) The error function dropping during a simulation.

16.3 Setting the variables to fit

In figure 16.4, 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. The tab entitled *Fit variables* 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 16.6). You can see from Figure 16.6, that there are 7 columns to the fit window, they have the following functions:

- Enabled: This enables or disables the fitting of that variable.
- Variable: This describes the path of the variable to be fit in *English*.
- Min: This is the minimum value the variable can take.
- Max: This is the maximum value the variable can take.
- Error function: If the variable strays out of the min-max range, this number will be added onto the total fitting error. This idea is this *scares* the algorithm back into the allowed range.
- Log scale: This determines if the variable is fit on a log scale. For parameters which range over many orders of magnitude it is often useful to make sure the algorithm can explore the entire range.
- Variable (json): This is usually hidden, but represents the full path of the parameter to be fit in *json* format, it is this and not the *English* path that is used by the back end. The *English* path is generated for humans and can indeed not be correct as long as the json path is correct.

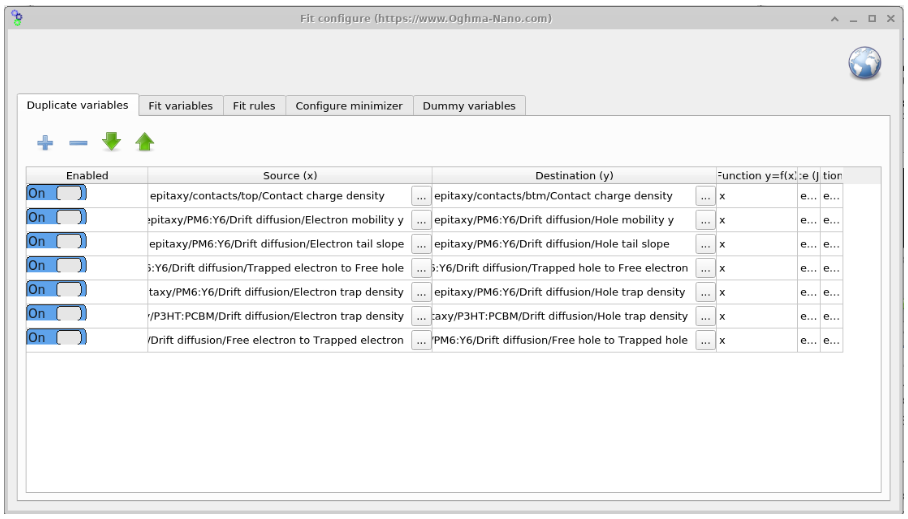


Enabled	Variable		Min	Max	Error function	Log scale	Var
On	epitaxy/contacts/top/Contact charge density	...	100000000...	1e+27	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Electron mobility y	...	1e-10	0.0001	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Hole mobility y	...	1e-10	0.0001	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Electron tail slope	...	0.03	0.15	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Hole tail slope	...	0.03	0.15	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Electron trap density	...	100000000...	1e+25	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Hole trap density	...	100000000...	1e+25	100.0	Off	e...
Off	PM6:Y6/Drift diffusion/Free electron to Trapped electron	...	1e-25	1e-20	100.0	Off	e...
Off	axy/PM6:Y6/Drift diffusion/Trapped electron to Free hole	...	1e-25	1e-20	100.0	Off	e...
Off	axy/PM6:Y6/Drift diffusion/Trapped hole to Free electron	...	1e-25	1e-20	100.0	Off	e...
Off	epitaxy/PM6:Y6/Drift diffusion/Free hole to Trapped hole	...	1e-25	1e-20	100.0	Off	e...
On	parasitic/Shunt resistance	...	0.001	1000000.0	100.0	Off	...
Off	parasitic/Series resistance	...	1.0	200.0	100.0	Off	...
Off	ısion/n_{free} to p_{free} Recombination rate constant	...	0.0	1e-10	100.0	Off	e...

Figure 16.6: Defining the variables to fit and their ranges.

16.4 Duplicating variables

Also in the *Fit variable window* is a tab called *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 (as a variable) but we are using the *Duplicate 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 16.7).



Enabled	Source (x)		Destination (y)		Function y=f(x)	:e (J	tion
On	epitaxy/contacts/top/Contact charge density	...	epitaxy/contacts/btm/Contact charge density	...	x	e...	e...
On	epitaxy/PM6:Y6/Drift diffusion/Electron mobility y	...	epitaxy/PM6:Y6/Drift diffusion/Hole mobility y	...	x	e...	e...
On	epitaxy/PM6:Y6/Drift diffusion/Electron tail slope	...	epitaxy/PM6:Y6/Drift diffusion/Hole tail slope	...	x	e...	e...
On	5:Y6/Drift diffusion/Trapped electron to Free hole	...	5:Y6/Drift diffusion/Trapped hole to Free electron	...	x	e...	e...
On	taxy/PM6:Y6/Drift diffusion/Electron trap density	...	epitaxy/PM6:Y6/Drift diffusion/Hole trap density	...	x	e...	e...
On	/P3HT:PCBM/Drift diffusion/Electron trap density	...	taxy/P3HT:PCBM/Drift diffusion/Hole trap density	...	x	e...	e...
On	/Drift diffusion/Free electron to Trapped electron	...	PM6:Y6/Drift diffusion/Free hole to Trapped hole	...	x	e...	e...

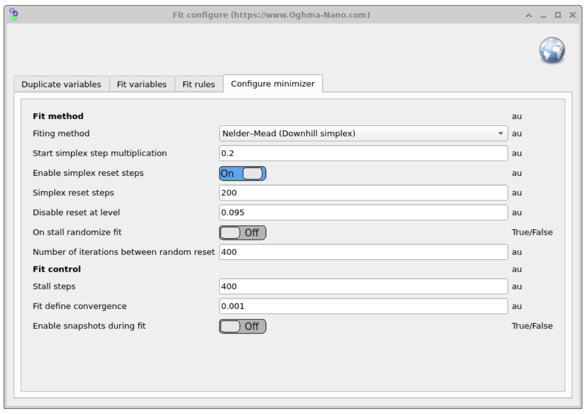
Figure 16.7: Defining which variables are duplicated to other variables.

16.5 Fit rules

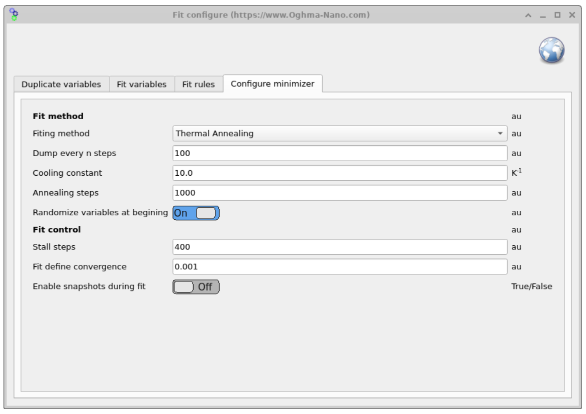
Also in the *Fit variable window* is a tab called *Fit rules.* This is used to enforce rules onto the fit, using mathematical expressions. For example one can force one parameter to always be bigger than another.

16.6 Minimizer configuration

Also in the *Fit configure* window is a tab called *Configure minimizer*. This configures the fitting algorithm that is used for the fit. Fitting algorithms are an entire research field in themselves. OghmaNano implements a few key algorithms to do the fitting. These are listed below. The algorithm used can be selected from the *Fitting method* drop down box.



Fit method		au
Fiting method	Nelder–Mead (Downhill simplex)	au
Start simplex step multiplication	0.2	au
Enable simplex reset steps	On	au
Simplex reset steps	200	au
Disable reset at level	0.095	au
On stall randomize fit	Off	True/False
Number of iterations between random reset	400	au
Fit control		au
Stall steps	400	au
Fit define convergence	0.001	au
Enable snapshots during fit	Off	True/False



Fit method		au
Fiting method	Thermal Annealing	au
Dump every n steps	100	au
Cooling constant	10.0	K^{-1}
Annealing steps	1000	au
Randomize variables at begining	On	au
Fit control		au
Stall steps	400	au
Fit define convergence	0.001	au
Enable snapshots during fit	Off	True/False

Figure 16.8: a) Configuring the simplex downhill minimizer b) Configuring the thermal annealing minimizer.

16.6.1 Nelder-Mead (Simplex Downhill)

The simplex downhill algorithm is the most often used fitting method in OghmaNano, all papers published until 2024 used this method. More can be read about how the algroytham works here: `https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method`. This minimizer has the following options:

- 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.
- Enable snapshots during fit: By default snapshots are turned off during fitting as they produce a lot of disk access to allow them the 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.

The advantage of this method is that it effectively looks for an answer by rolling a ball down hill. It also does not need to take gradients of the problem, which can be beneficial in complex problem spaces.

16.6.2 Thermal Annealing - experimental

This is a classical thermal annealing algorithm. The algorithm will only explore the parameter space defined by the allowed limits of the fit variables, thus to use this efficiently it is key to set these limits correctly. The method is good for small simple problems.

- Dump every n steps: Writes the output to disk every n steps
- Cooling constant: Defines how quickly the problem cools, the cooling takes the form of $e^{-k(T_{start}-T_{stop})}$, where T_{start} is 300K and T_{stop} is 0K.
- Annealing steps: Defines the number of steps taken to get from T_{start} to T_{stop}.

The advantage of this method is that it is very simple.

16.6.3 Markov chain Monte Carlo (MCMC)

This method is under development.

16.6.4 Hamiltonian Monte Carlo (HMC)

This method is under development.

No-U-Turn Sampler (NUTS)

This method is under development.

16.6.5 Newton

This method is under development.

This is a gradient method, thus requires derivatives to be calculated. This can make the fitting process unstable, but the advantage is that it can be faster than Nelder-Mead.

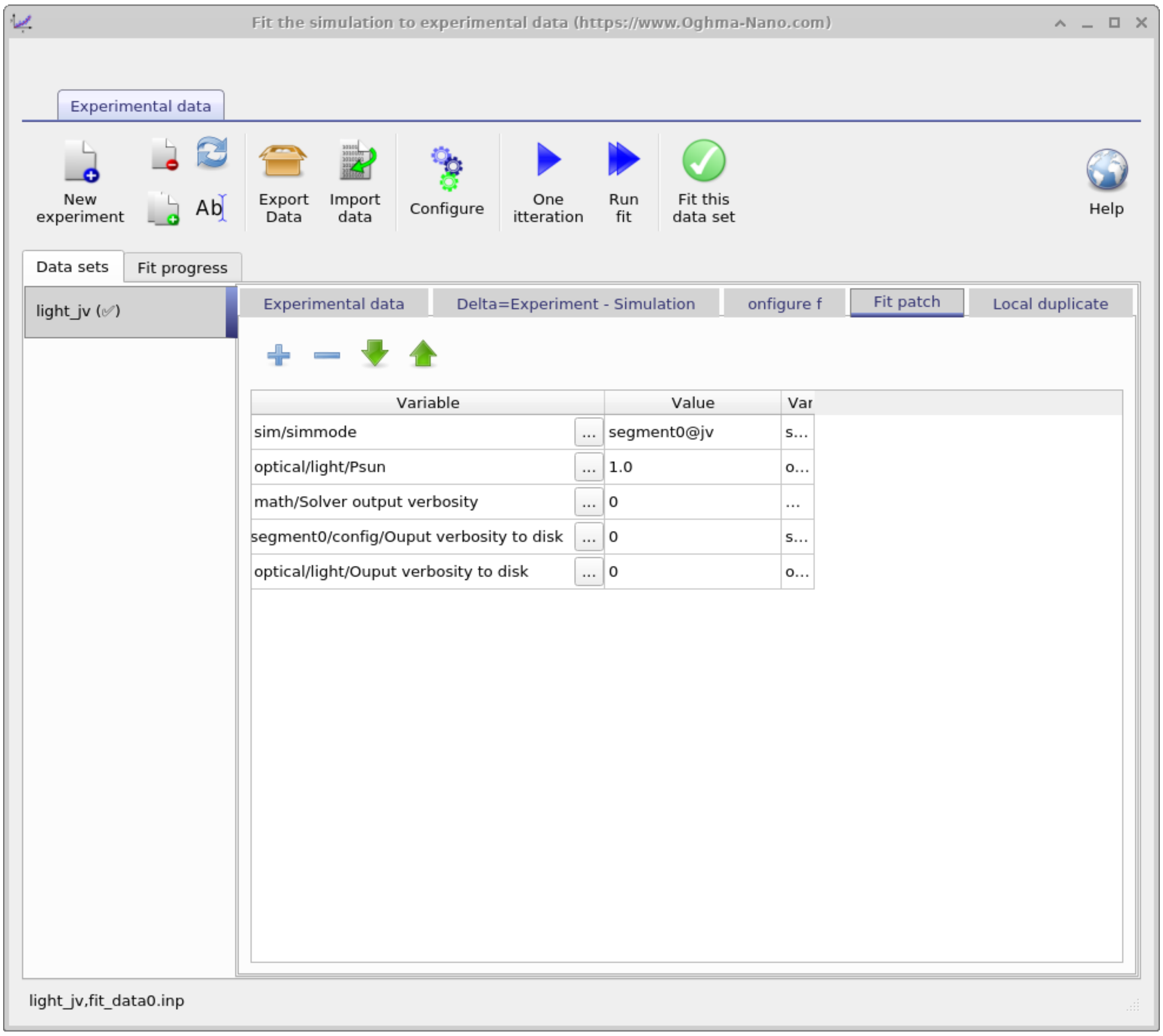
16.6.6 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

This method is under development.

This is a gradient method, thus requires derivatives to be calculated. This can make the fitting process unstable, but the advantage is that it can be faster than Nelder-Mead.

16.7 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 16.9) 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.



Variable	Value	Var
sim/simmode	segment0@jv	s...
optical/light/Psun	1.0	o...
math/Solver output verbosity	0	...
segment0/config/Ouput verbosity to disk	0	s...
optical/light/Ouput verbosity to disk	0	o...

Figure 16.9: The fit patch applied to each data set.

16.8 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/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 them 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 17

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 OghamNano simulations:

1. The first method is using the parameter scan window, this is described below in section 17.1. The parameter scan window allows a user to vary a parameter (or multiple paramters) 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 17.3.1. Python scripting allws 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 17.3.2. The advantage of MATLAB scripting is that lots of people can code in MATLAB so makes automating OghmaNano very accessable. 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 systematic 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 14.1) 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...) the you can automate OghmaNano.

17.1 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 17.1). 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 17.2. This will bring up the scan window, see figure 17.3.

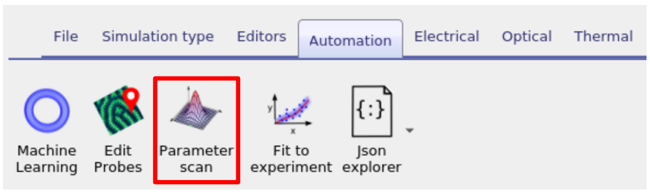


Figure 17.1: Step 1: Select the Parameter scan tool, to bring up the parameter scan window.

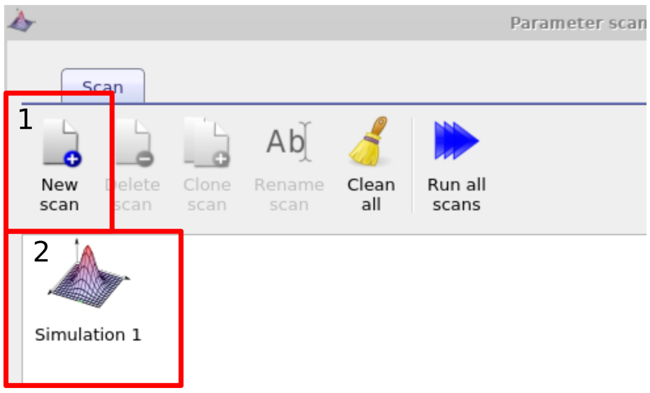
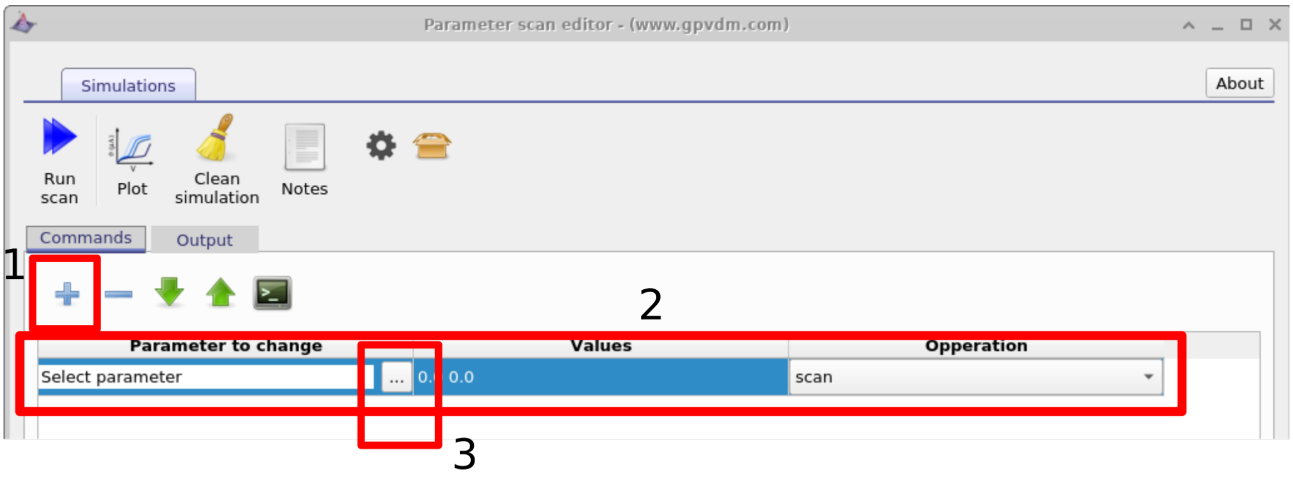


Figure 17.2: Step 2: Make a new parameter scan, then double click on it to open it.

17.1.1 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 17.3, 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.



Parameter to change		Values	Opperation
Select parameter	...	0. 0.0	scan

Figure 17.3: 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 17.4. This file is a tree structure, see 14.1. 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*.

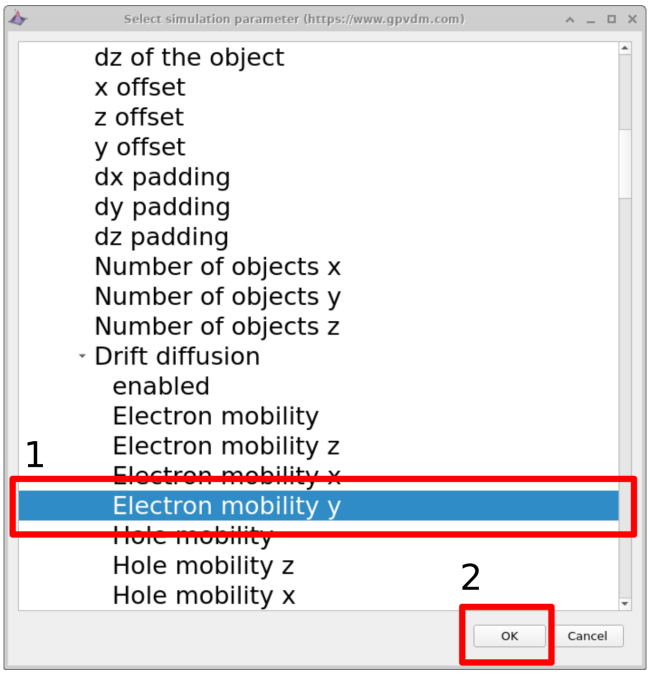
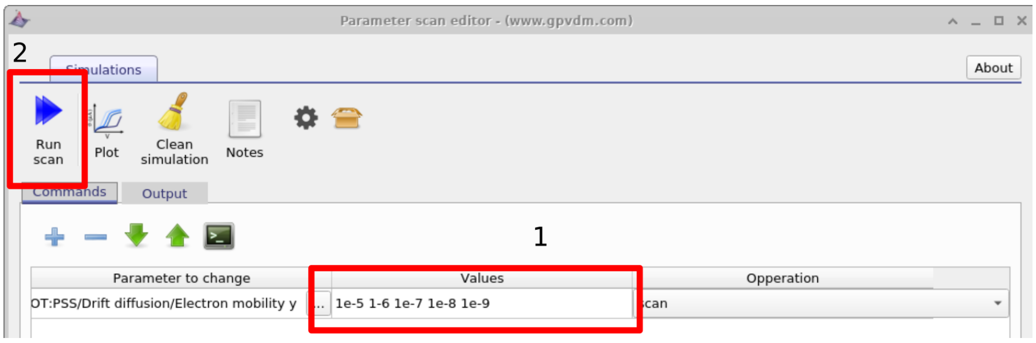


Figure 17.4: Step 5: Select the parameter you want to scan in the parameter selection window, in this case we are selecting epitaxy→ PM6:Y6→ Drift diffusion→ Electron mobility y.

- 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*.

Next enter the values of mobility which you want to scan over in this case we will be entering *1e-5 1-6 1e-7 1e-8 1e-9* (see figure 17.5 1) then click *run scan* (see figure 17.5 2). OghmaNano will run one simulation on each core of your computer until all the simulations are finished.



Parameter to change	Values	Opperation
OT:PSS/Drift diffusion/Electron mobility y	1e-5 1-6 1e-7 1e-8 1e-9	can

Figure 17.5: 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 17.6. 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 17.6. OghmaNano will automaticity plot all the curves from each simulation in one graph, see figure 17.7.

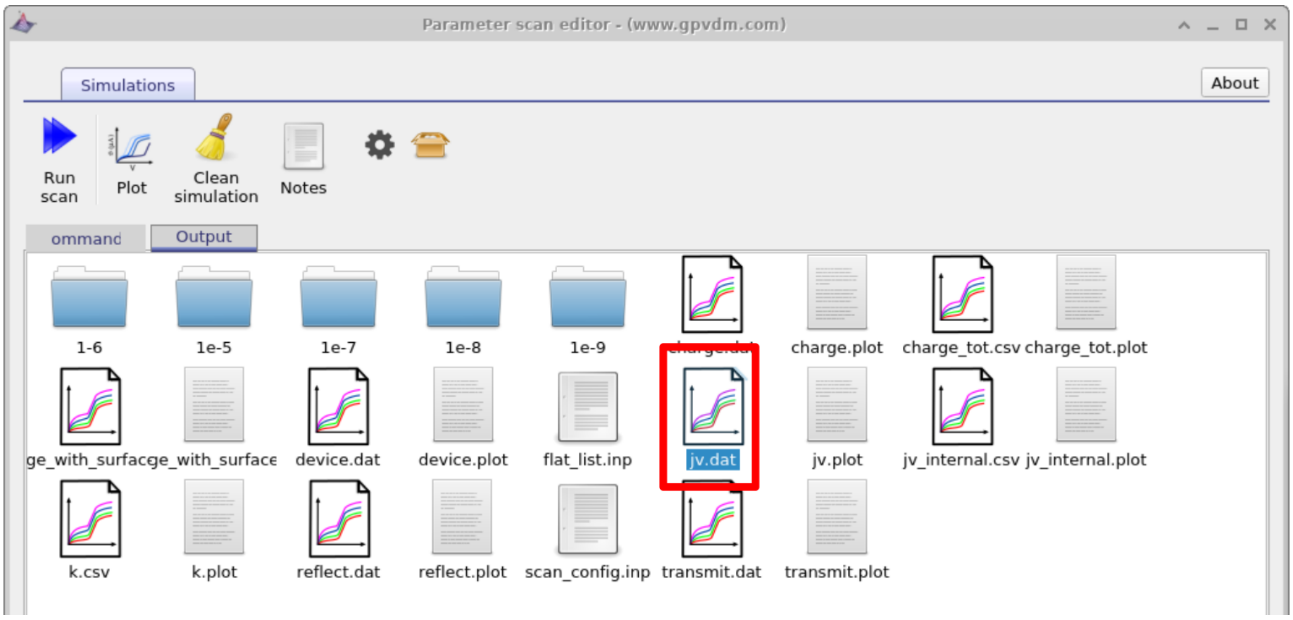


Figure 17.6: Step 7: The output tab showing the five simulation directories and the multicolored plot files.

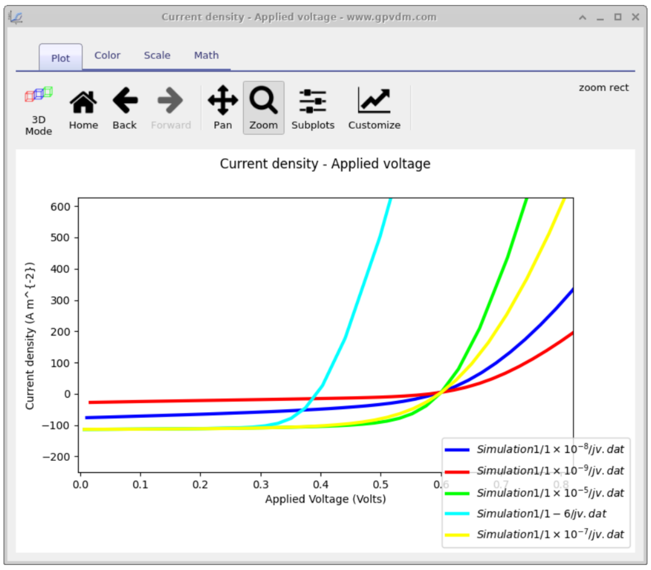


Figure 17.7: Step 8: The result of the mobility scan.

17.1.2 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 17.8. 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 8 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

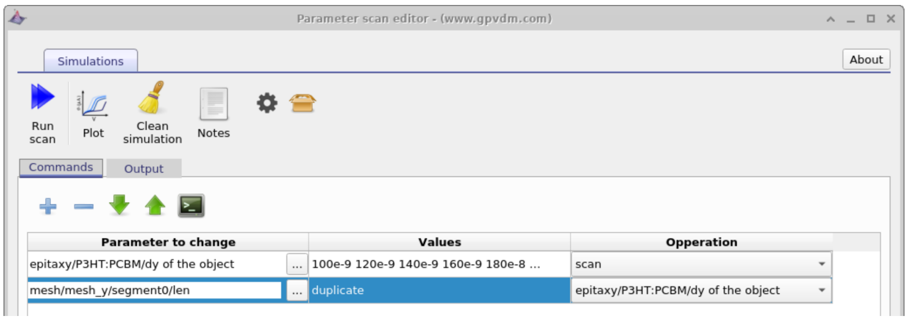
mesh→mesh_y→segment0→len

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.



Parameter to change	Values	Opperation
epitaxy/P3HT:PCBM/dy of the object	100e-9 120e-9 140e-9 160e-9 180e-8 ...	scan
mesh/mesh_y/segment0/len	duplicate	epitaxy/P3HT:PCBM/dy of the object

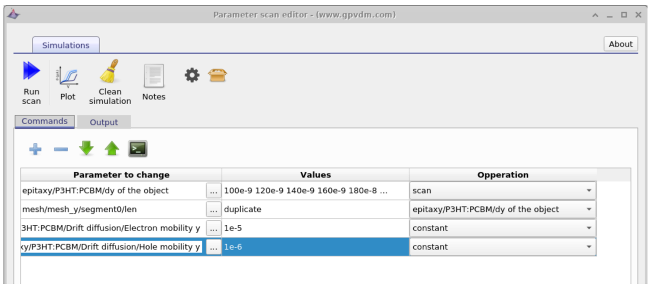
Figure 17.8: Duplicating material paramters.

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 8). 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 17.8 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.

17.1.3 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 17.9



Parameter to change	Values	Opperation
epitaxy/P3HT:PCBM/dy of the object	100e-9 120e-9 140e-9 160e-9 180e-8 ...	scan
mesh/mesh_y/segment0/len	duplicate	epitaxy/P3HT:PCBM/dy of the object
3HT:PCBM/Drift diffusion/Electron mobility y	1e-5	constant
xy/P3HT:PCBM/Drift diffusion/Hole mobility y	1e-6	constant

Figure 17.9: The result of the mobility scan.

17.1.4 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.

17.1.5 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 were 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.

17.2 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.

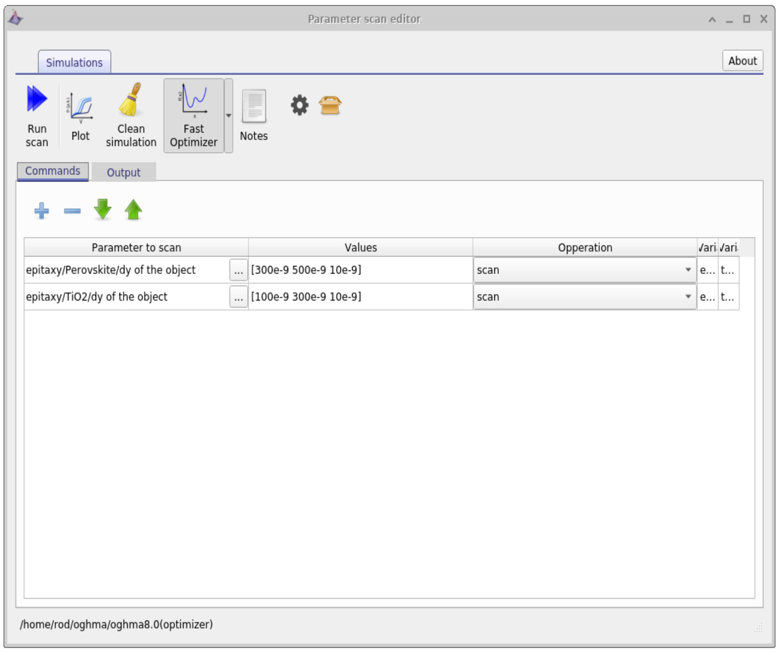
Very often when optimizing a device an engineer or scientists will be want to know what the optimum structure of a device is. For example a perovskite solar cell is made up of multiple layers, but what is the optimum thickness of each layer? If the perovskite layer is made really thick then lots of light 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 pattens, 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.

17.2.1 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 17.10. 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



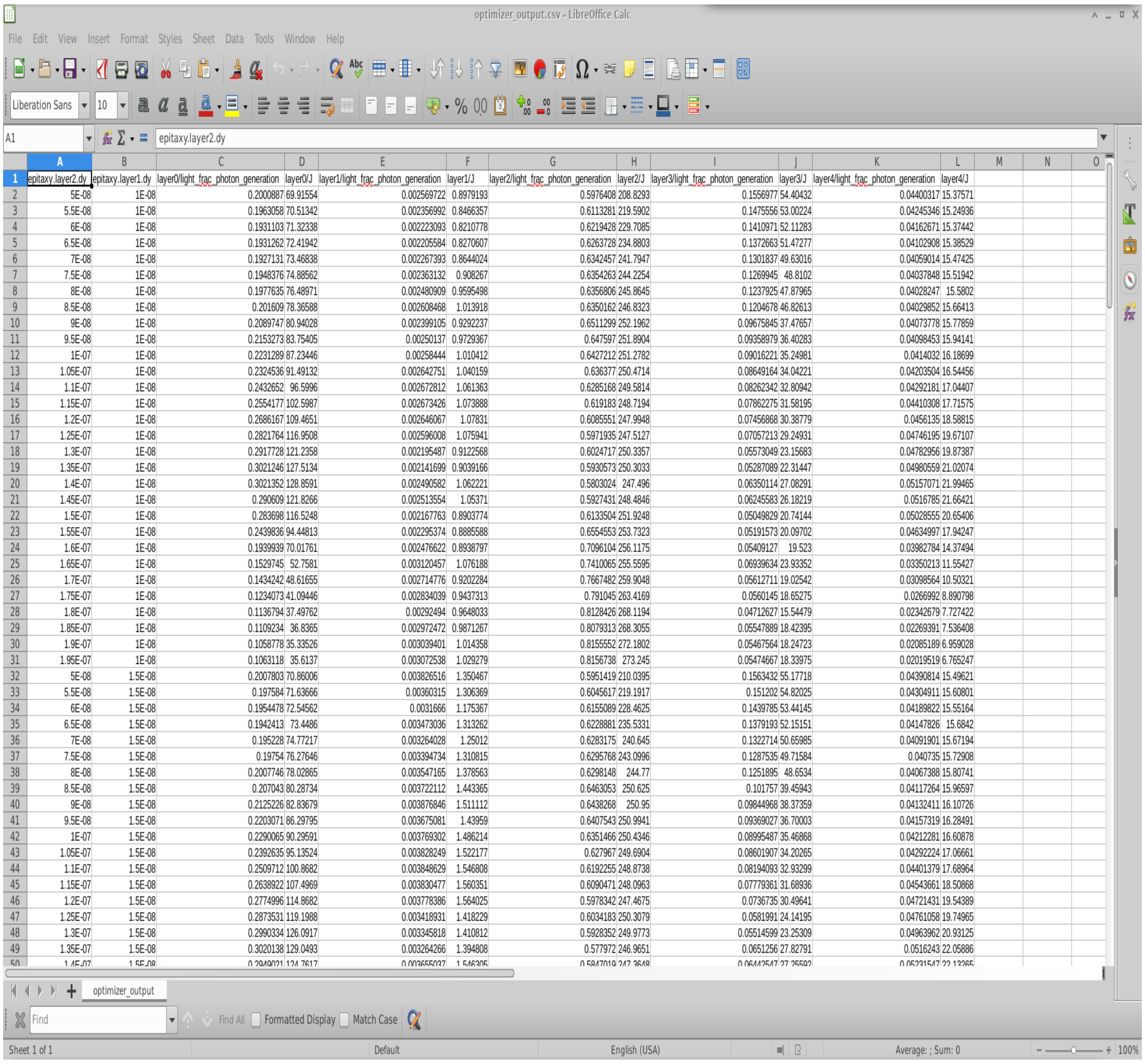
Parameter to scan	Values	Opperation	Vari	Vari
epitaxy/Perovskite/dy of the object	[300e-9 500e-9 10e-9]	scan	e...	t...
epitaxy/TiO2/dy of the object	[100e-9 300e-9 10e-9]	scan	e...	t...

/home/rod/oghma/oghma8.0(optimizer)

Figure 17.10: 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 17.11.

If you examine figure 17.11 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 with in 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.



	A	B	C	D	E	F	G	H	I	J	K	L
1	epitaxy.layer2.dy	epitaxy.layer1.dy	layer0/light_frac_photon_generation	layer0/J	layer1/light_frac_photon_generation	layer1/J	layer2/light_frac_photon_generation	layer2/J	layer3/light_frac_photon_generation	layer3/J	layer4/light_frac_photon_generation	layer4/J
2	5E-08	1E-08	0.2000887	69.91554	0.002569722	0.8979193	0.5976408	208.8293	0.1556977	54.40432	0.04400317	15.37571
3	5.5E-08	1E-08	0.1963058	70.51342	0.002356992	0.8466357	0.6113281	219.5902	0.1475556	53.00224	0.04245346	15.24936
4	6E-08	1E-08	0.1931103	71.32338	0.002223093	0.8210778	0.6219428	229.7085	0.1410971	52.11283	0.04162671	15.37442
5	6.5E-08	1E-08	0.1931262	72.41942	0.002205584	0.8270607	0.6263728	234.8803	0.1372663	51.47277	0.04102908	15.38529
6	7E-08	1E-08	0.1927131	73.46838	0.002267393	0.8644024	0.6342457	241.7947	0.1301837	49.63016	0.04059014	15.47425
7	7.5E-08	1E-08	0.1948376	74.88562	0.002363132	0.908267	0.6354263	244.2254	0.1269945	48.8102	0.04037848	15.51942
8	8E-08	1E-08	0.1977635	76.48971	0.002480909	0.9595498	0.6356806	245.8645	0.1237925	47.87965	0.04028247	15.5802
9	8.5E-08	1E-08	0.201609	78.36588	0.002608468	1.013918	0.6350162	246.8323	0.1204678	46.82613	0.04029852	15.66413
10	9E-08	1E-08	0.2089747	80.94028	0.002399105	0.9292237	0.6511299	252.1962	0.09675845	37.47657	0.04073778	15.77859
11	9.5E-08	1E-08	0.2153273	83.75405	0.00250137	0.9729367	0.647597	251.8904	0.09358979	36.40283	0.04098453	15.94141
12	1E-07	1E-08	0.2231289	87.23446	0.00258444	1.010412	0.6427212	251.2782	0.09016221	35.24981	0.0414032	16.18699
13	1.05E-07	1E-08	0.2324536	91.49132	0.002642751	1.040159	0.636377	250.4714	0.08649164	34.04221	0.04203504	16.54456
14	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
15	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
16	1.2E-07	1E-08	0.2686167	109.4651	0.002646067	1.07831	0.6085551	247.9948	0.07456868	30.38779	0.0456135	18.58815
17	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
18	1.3E-07	1E-08	0.2917728	121.2358	0.002195487	0.9122568	0.6024717	250.3357	0.05573049	23.15683	0.04782956	19.87387
19	1.35E-07	1E-08	0.3021246	127.5134	0.002141699	0.9039166	0.5930573	250.3033	0.05287089	22.31447	0.04980559	21.02074
20	1.4E-07	1E-08	0.3021352	128.8591	0.002490582	1.062221	0.5803024	247.496	0.06350114	27.08291	0.05157071	21.99465
21	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
22	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
23	1.55E-07	1E-08	0.2439836	94.44813	0.002295374	0.8885588	0.6554553	253.7323	0.05191573	20.09702	0.04634997	17.94247
24	1.6E-07	1E-08	0.1939939	70.01761	0.002476622	0.8938797	0.7096104	256.1175	0.05409127	19.523	0.03982784	14.37494
25	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
26	1.7E-07	1E-08	0.1434242	48.61655	0.002714776	0.9202284	0.7667482	259.9048	0.05612711	19.02542	0.03098564	10.50321
27	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
28	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
29	1.85E-07	1E-08	0.1109234	36.8365	0.002972472	0.9871267	0.8079313	268.3055	0.05547889	18.42395	0.02269391	7.536408
30	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
31	1.95E-07	1E-08	0.1063118	35.6137	0.003072538	1.029279	0.8156738	273.245	0.05474667	18.33975	0.02019519	6.765247
32	5E-08	1.5E-08	0.2007803	70.86006	0.003826516	1.350467	0.5951419	210.0395	0.1563432	55.17718	0.04390814	15.49621
33	5.5E-08	1.5E-08	0.197584	71.63666	0.00360315	1.306369	0.6045617	219.1917	0.151202	54.82025	0.04304911	15.60801
34	6E-08	1.5E-08	0.1954478	72.54562	0.0031666	1.175367	0.6155089	228.4625	0.1439785	53.44145	0.04189822	15.55164
35	6.5E-08	1.5E-08	0.1942413	73.4486	0.003473036	1.313262	0.6228881	235.5331	0.1379193	52.15151	0.04147826	15.6842
36	7E-08	1.5E-08	0.195228	74.77217	0.003264028	1.25012	0.6283175	240.645	0.1322714	50.65985	0.04091901	15.67194
37	7.5E-08	1.5E-08	0.19754	76.27646	0.003394734	1.310815	0.6295768	243.0996	0.1287535	49.71584	0.040735	15.72908
38	8E-08	1.5E-08	0.2007746	78.02865	0.003547165	1.378563	0.6298148	244.77	0.1251895	48.6534	0.04067388	15.80741
39	8.5E-08	1.5E-08	0.207043	80.28734	0.003722112	1.443365	0.6463053	250.625	0.101757	39.45943	0.04117264	15.96597
40	9E-08	1.5E-08	0.2125226	82.83679	0.003876846	1.511112	0.6438268	250.95	0.09844968	38.37359	0.04132411	16.10726
41	9.5E-08	1.5E-08	0.2203071	86.29795	0.003675081	1.43959	0.6407543	250.9941	0.09369027	36.70003	0.04157319	16.28491
42	1E-07	1.5E-08	0.2290065	90.29591	0.003769302	1.486214	0.6351466	250.4346	0.08995487	35.46868	0.04212281	16.60878
43	1.05E-07	1.5E-08	0.2392635	95.13524	0.003828249	1.522177	0.627967	249.6904	0.08601907	34.20265	0.04292224	17.06661
44	1.1E-07	1.5E-08	0.2509712	100.8682	0.003848629	1.546808	0.6192255	248.8738	0.08194093	32.93299	0.04401379	17.68964
45	1.15E-07	1.5E-08	0.2638922	107.4969	0.003830477	1.560351	0.6090471	248.0963	0.07779361	31.68936	0.04543661	18.50868
46	1.2E-07	1.5E-08	0.2774996	114.8682	0.003778386	1.564025	0.5978342	247.4675	0.0736735	30.49641	0.04721431	19.54389
47	1.25E-07	1.5E-08	0.2873531	119.1988	0.003418931	1.418229	0.6034183	250.3079	0.0581991	24.14195	0.04761058	19.74965
48	1.3E-07	1.5E-08	0.2990334	126.0917	0.003345818	1.410812	0.5928352	249.9773	0.05514599	23.25309	0.04963962	20.93125
49	1.35E-07	1.5E-08	0.3020138	129.0493	0.003264266	1.394808	0.577972	246.9651	0.0651256	27.82791	0.0516243	22.05886
50	1.4E-07	1.5E-08	0.2949021	124.7617	0.003655037	1.546305	0.5847019	247.3648	0.06442547	27.25592	0.05231547	22.13265

Figure 17.11: The file your simulation directory/optimizer/optimizer_output.csv opened in LibreOffice (You can use Excel).

17.3 Python/MATLAB scripting of OghmaNano

Scripting offers a more powerful way to interact with gvpdm. 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)`

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

17.3.1 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 14.1, .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 gpvmd 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.

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.

```python
#!/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.

17.3.2 MATLAB scripting

As described in section 14.1 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.

```python
#!/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 GNUPLOT compatible files for plotting the results together.
api.build_multiplot(scan_dir,gnuplot=True])
```

Listing 7: Running jobs across multiple CPUs using python

```
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
origonal_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(origonal_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(origonal_path,"sim.oghma"),\\
        fullfile(origonal_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(origonal_path)
```

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

Chapter 18

Machine learning

This chapter overs generating machine learning datasets using OghmaNano. Typically you would generate these datasets then use external software such as https://www.tensorflow.org to perform the learning/prediction.

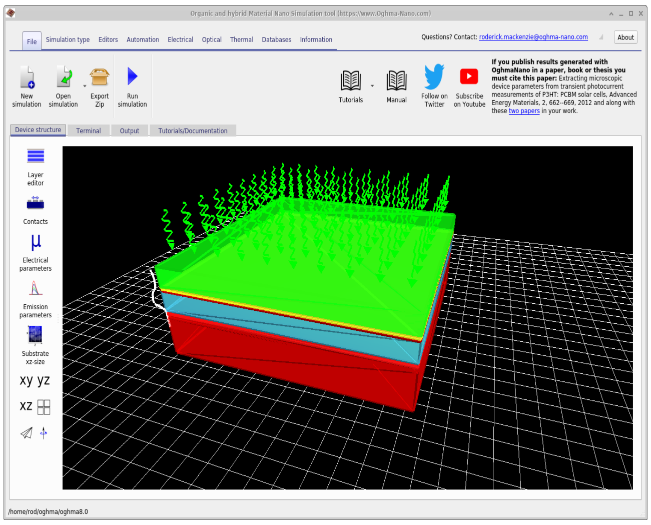
18.1 Generating ML datasets

18.1.1 Introduction

One will often want to extract physical device parameters form a set of data using modelling. For example you may have a set of JV curves and what to extract the charge carrier mobility and recombination rate for that device. The traditional way of doing this would be to fit a model such as OghmaNano to the data set (This is described in section 16). The drawback of this approach is that it can take a significant amount of time to fit one data set, furthermore when one has multiple data sets the challenge can be significant. The fitting process is complex and requires someone who is an expert in simulation with a lot of patience to perform it. This is the reason why fitting of device models to experimental data is only performed by a small subset of the community.

A more modern approach to this problem is to use machine learning. With this approach rather than trying to fit a single JV curve, one will set up a simulation representing the device structure. However, rather than fitting the individual electrical parameters to the data set, one will generate many thousands copies of that one device but with randomly chosen electrical parameters. Each one of these devices is referred to as a *virtual device.* A simulation program such as OghmaNano will then be used to generate JV curves for each of these virtual devices. Thus the user will have JV curves and the corresponding electrical parameters for each *virtual device.* This data set can then be used to train a machine learning model to predict the electrical parameters from JV curves. Thus, OghmaNano is acting as a forward transform, transforming electrical parameters to simulated experimental data, then the machine learning is acting as the reverse transform that can reverse the simulation process. Once trained this model can then be used to extract material paramters from real devices.

Advantages of this approach is that once the machine learning algorithm is trained it can extract material parameters within seconds from a device and the user does not need to be an expert at simulation to use the trained network. However, for this method to work is key to generate a large data set on which to train the machine learning models. This process is described in the following pages.


Organic and hybrid Material Nano Simulation tool (https://www.Oghma-Nano.com)

Questions? Contact: roderick.mackenzie@oghma-nano.com

If you publish results generated with OghmaNano in a paper, book or thesis you must cite this paper: Extracting microscopic device parameters from transient photocurrent measurements of P3HT: PCBM solar cells, Advanced Energy Materials, 2, 662--669, 2012 and along with these two papers in your work.

/home/rod/oghma8.0

Figure 18.1: The main simulation window showing the *Automation* ribbon, within this ribbon the *Machine learning* icon can be seen.

18.1.2 Generating a machine learning dataset

This section assumes you have correctly configured your device layer structure in the layer editor. However, if you want to start with a preconfigured simulation you will be able to find this example in the New Simulation window under the *ML Example*. Figure 18.1 shows the main window for a configured device, in the automation ribbon you will see a *Machine learning* icon which looks like a round blue circle. If you click on this then it will open the main machine learning window show in Figure 18.2.

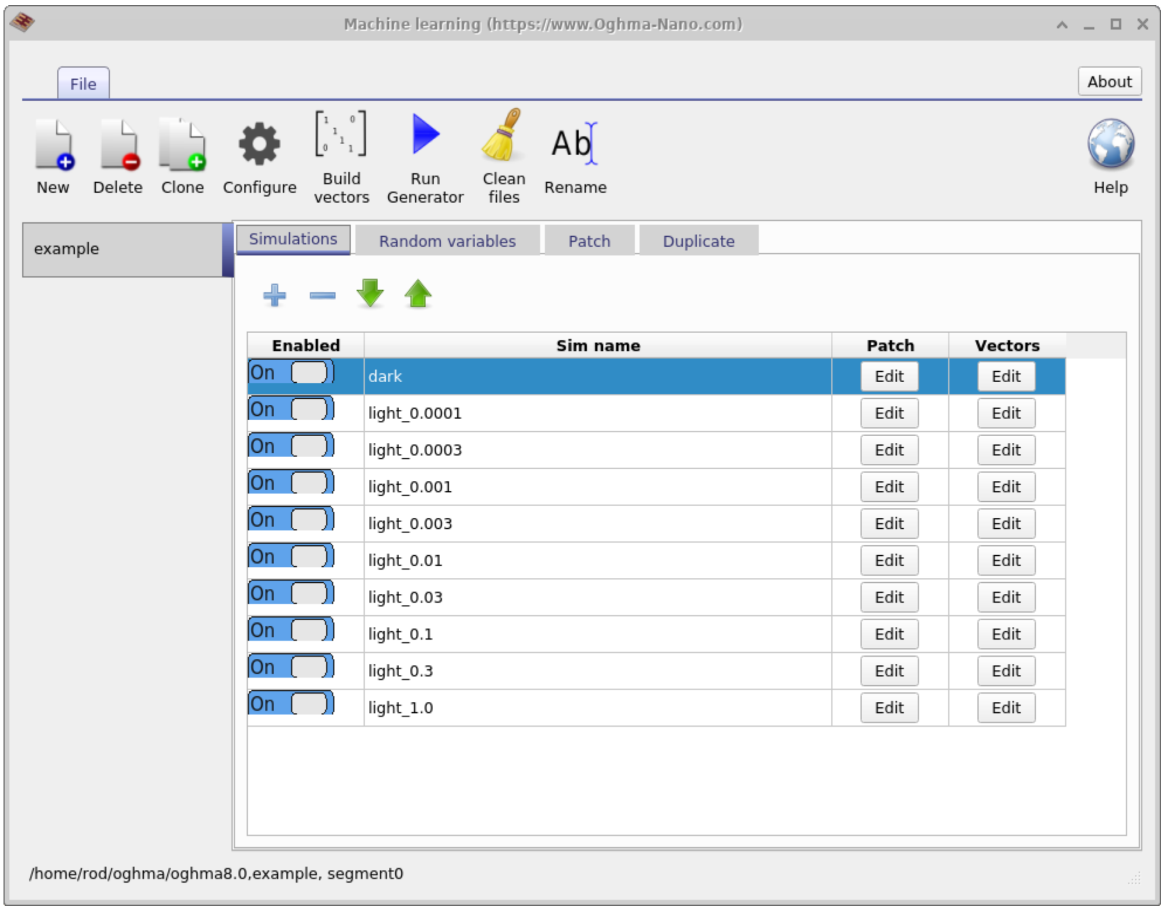
The main purpose of this window is to build large data sets for training machine learning algorithms. The data set we are generating is called *example*. The first tab that you can see open is called *Simulations*. In this tab one defines the simulations you want performed for each *virtual device*. In this case we will simulate a dark curve and 9 light curves varying between 0.0001 Suns and 1.0 Suns. The simulations can be disabled by setting *Enabled* to *off*. If one clicks on the the *Edit* button for the line *light*_0.0001 in the *Patch* column, the *Patch window* shown in Figure 18.3 will appear.

The purpose of the *Patch window* is to change parameters in the virtual simulation. For example in our case we which to simulate a dark JV curve and nine different light intensities, therefore for each of these simulation the light intensity must be correctly adjusted. If one opens the patch window for *light*_0.0001, you will see that the simulation variable *optical/light/Psun* which controls the light intensity is set to 0.0001. If one opens the patch window for dark, one will see that the value of *optical/light/Psun* is set to 0.0. In this way using the same base simulation multiple experiments under different conditions can be simulated.

If one now clicks on the *Edit button* in the *Vectors* column for the line *light*_0.0001 in Figure 18.2 the vectors window will appear, this is shown in Figure 18.4.

The vectors window is used to define what data is extracted from the finished simulation to act as a machine learning input vector. In this case we are extracting data points between -2.0 and 1.4 V from the file *jv.dat*, which contains the current voltage curve. If one were simulating a time domain measurement one may wish to extract various time points from the *time_i.csv* file or any other appropriate file. The number of points chosen to form the vector is arbitrary and up to the user. A longer input vector will make the machine learning process slower, but potentially capture more features of the JV curve.

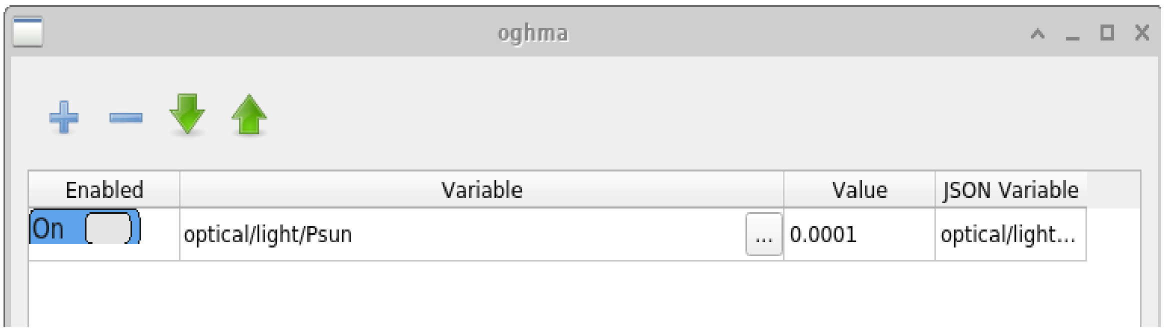
The next step is to set up which variables one wishes to randomize, this is done in the *Random variables* tab of the main machine learning window. The window has five columns; the first column (Enabled) dictates if the random variable is used; the second column (Variable)



Enabled	Sim name	Patch	Vectors
On	dark	Edit	Edit
On	light_0.0001	Edit	Edit
On	light_0.0003	Edit	Edit
On	light_0.001	Edit	Edit
On	light_0.003	Edit	Edit
On	light_0.01	Edit	Edit
On	light_0.03	Edit	Edit
On	light_0.1	Edit	Edit
On	light_0.3	Edit	Edit
On	light_1.0	Edit	Edit

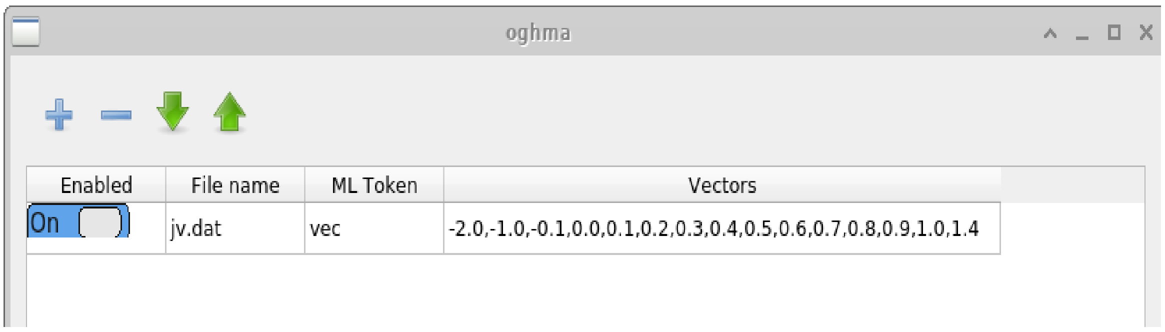
/home/rod/oghma/oghma8.0,example, segment0

Figure 18.2: The main machine learning window.



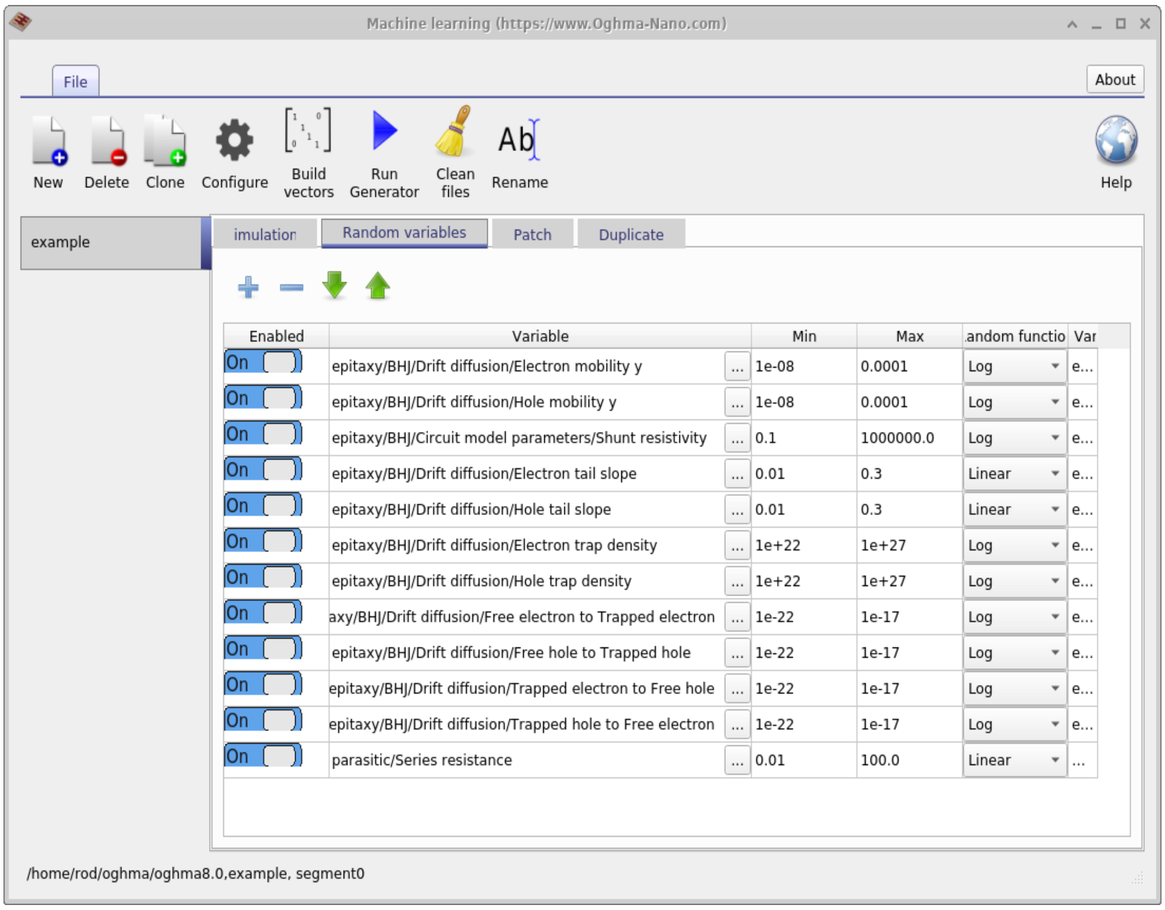
Enabled	Variable	Value	JSON Variable
On	optical/light/Psun ...	0.0001	optical/light...

Figure 18.3: The simulation patch window, this window is used to change a value in a sub simulation, in this example we are changing the light intensity.



Enabled	File name	ML Token	Vectors
On	jv.dat	vec	-2.0,-1.0,-0.1,0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.4

Figure 18.4: The vectors window, this window defines what data is extracted from the simulation as a machine learning vector.



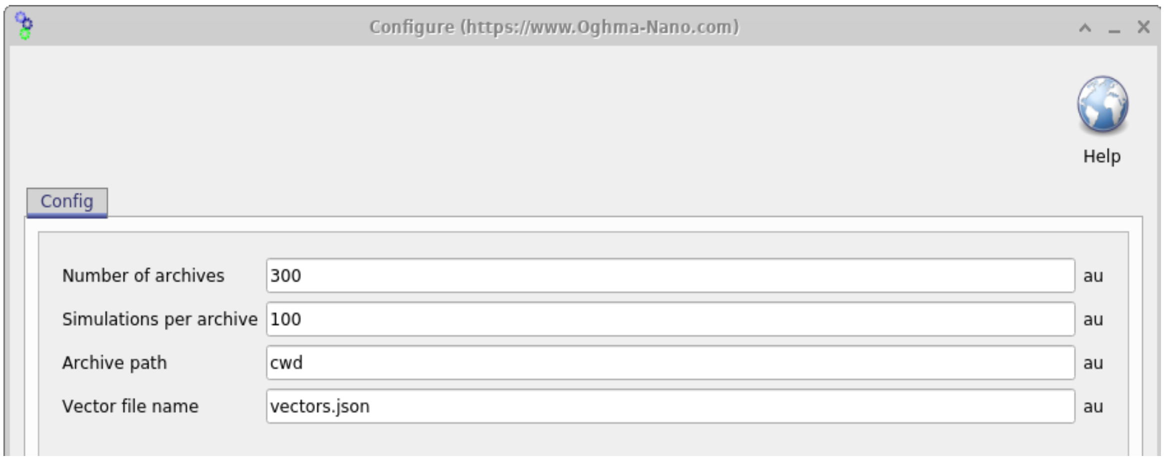
Enabled	Variable	Min	Max	andom functio	Var
On	epitaxy/BHJ/Drift diffusion/Electron mobility y ...	1e-08	0.0001	Log	e...
On	epitaxy/BHJ/Drift diffusion/Hole mobility y ...	1e-08	0.0001	Log	e...
On	epitaxy/BHJ/Circuit model parameters/Shunt resistivity ...	0.1	1000000.0	Log	e...
On	epitaxy/BHJ/Drift diffusion/Electron tail slope ...	0.01	0.3	Linear	e...
On	epitaxy/BHJ/Drift diffusion/Hole tail slope ...	0.01	0.3	Linear	e...
On	epitaxy/BHJ/Drift diffusion/Electron trap density ...	1e+22	1e+27	Log	e...
On	epitaxy/BHJ/Drift diffusion/Hole trap density ...	1e+22	1e+27	Log	e...
On	axy/BHJ/Drift diffusion/Free electron to Trapped electron ...	1e-22	1e-17	Log	e...
On	epitaxy/BHJ/Drift diffusion/Free hole to Trapped hole ...	1e-22	1e-17	Log	e...
On	epitaxy/BHJ/Drift diffusion/Trapped electron to Free hole ...	1e-22	1e-17	Log	e...
On	epitaxy/BHJ/Drift diffusion/Trapped hole to Free electron ...	1e-22	1e-17	Log	e...
On	parasitic/Series resistance ...	0.01	100.0	Linear	...

/home/rod/oghma/oghma8.0,example, segment0

Figure 18.5: The random variables that are used for the machine learning.

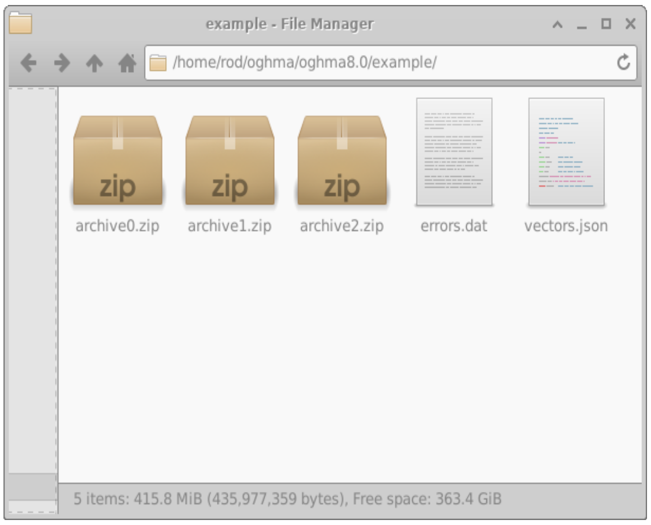
sets which variable is changed; the third column (Min) sets the minimum allowed value of the random value; the forth column (Max) sets the maximum allowed value of the random value; and the final column (Random function) dictates if the random variable should be chosen on a log or linear scale. Log scales are recommended for variables which span multiple orders of magnitude while linear scales are recommended if the variable spans around an order of magnitude. A log distribution will ensure that an evenly distributed span of numbers are chosen across the range when viewed on a log scale. An example of a linear parameter would be Urbach energy as it typically would vary from 30 to 150 meV and example of a log variable would be trap density as it typically can vary from $1 \times 10^{15} m^{-3} - 1 \times 10^{25} m^{-3}$.

Once the simulation has been set up, is is now time to consider how many virtual devices with randomized parameters one wishes to generate. If the *Settings window* is opened, see Figure 18.6 one can configure how many *virtual devices* are generated. This is controlled with two parameters, *Simulations per archive* (N_{sim}) and *Number of archives* (N_{arc}). These two numbers multiplied together gives you the total number of virtual devices generated. The generated simulation results are saved in zip files called *archives*, each archive will contain N_{sim} simulations. So in the example seen here, we will generate $N_{sim} * N_{arc}$ 3000 virtual devices stored across 100 zip files. It is important to note that for our example each virtual device will contain one dark JV curve simulation and 9 light simulations. Thus the number of files generated can quickly grow quite large. To prevent this being a problem the simulations are run in batches, with N_{sim} simulations being first generated then run by OghmaNano, then stored in a zip archive until N_{arc} archives are generated. In this way if the generation process is interrupted and one only looses the content of the archive currently being generated. Breaking up the data set like this also makes it easier to copy the files and more immune to corruption. Data generation will run across all CPU cores on your machine while archive generation will



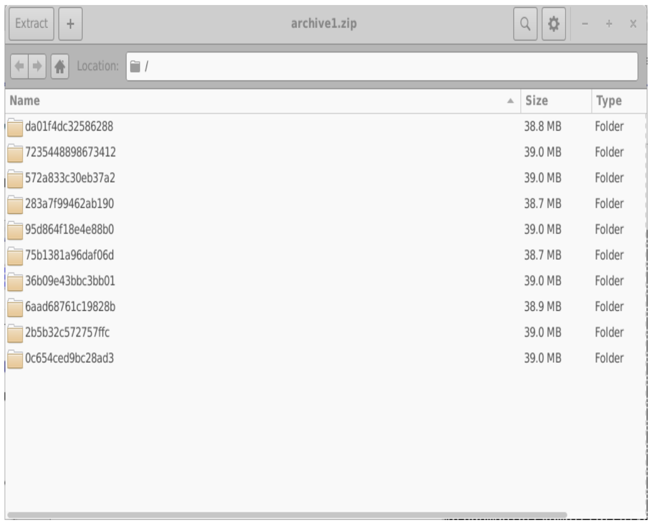
Number of archives	300	au
Simulations per archive	100	au
Archive path	cwd	au
Vector file name	vectors.json	au

Figure 18.6: Configuring the data generator.



archive0.zip archive1.zip archive2.zip errors.dat vectors.json

5 items: 415.8 MiB (435,977,359 bytes), Free space: 363.4 GiB



Name	Size	Type
da01f4dc32586288	38.8 MB	Folder
7235448898673412	39.0 MB	Folder
572a833c30eb37a2	39.0 MB	Folder
283a7f99462ab190	38.7 MB	Folder
95d864f18e4e88b0	39.0 MB	Folder
75b1381a96daf06d	38.7 MB	Folder
36b09e43bbc3bb01	39.0 MB	Folder
6aad68761c19828b	38.9 MB	Folder
2b5b32c572757ffc	39.0 MB	Folder
0c654ced9bc28ad3	39.0 MB	Folder

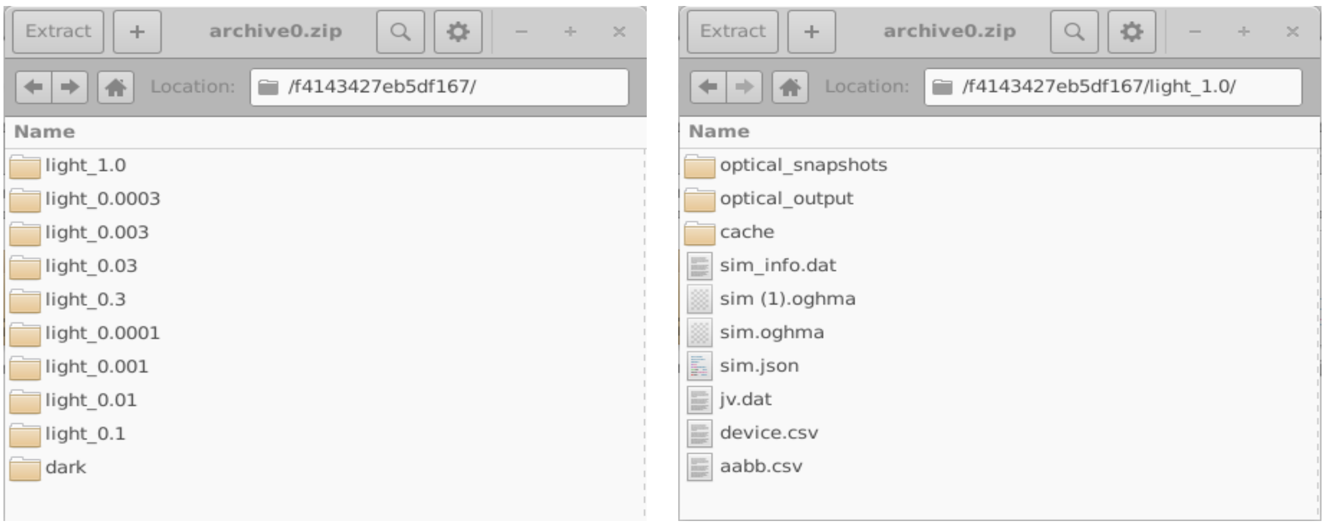
Figure 18.7: Left: The content of the *Example* directory; Right inside archive0.zip

only run on one core.

If now the *Run generator* button is pressed in the main Machine learning window Figure 18.2, OghamNano will start generating the simulations. After it is done, a directory called *example* will appear in your simulation directory, see the left of Figure 18.7. [Note to save time in this example *Simulations per archive* was set to 10 and *Number of archives* was set to 3]. The errors.dat file will contain any errors produced during the simulation. If one opens archive0.zip one will see the right of Figure 18.7.

Each of these directories is named with a random 16 digit hexadecimal number, each directory contains an entire set of simulations for one *virtual device*. So in our case each directory will contain one dark simulation and nine light simulations. The content of one of these directories is shown in the left of Figure 18.8. If one opens one of these files one can see Figure 18.8 right, one can see that it contains a full OghamNano simulation. There is sim.json file there and also the jv.dat file which contains the JV curve generated during the simulation. This directory contains some extra data including optical outputs and a cache file. It is recommended to try to minimize all output when generating these files otherwise the total size on disk becomes very large, this can be done by turning all the output verbosity options to produce minimal output.

In the above example we only have three archives but in a normal simulation run one would have up to 200 archives each with 200 simulations in each. Once the simulations have been performed one needs to convert these raw OghmaNano simulations into a vectors file for the machine learning algorithm. If one clicks on the *Build vectors* icon in the main machine learning



```
/f4143427eb5df167/
Name
light_1.0
light_0.0003
light_0.003
light_0.03
light_0.3
light_0.0001
light_0.001
light_0.01
light_0.1
dark
```

```
/f4143427eb5df167/light_1.0/
Name
optical_snapshots
optical_output
cache
sim_info.dat
sim (1).oghma
sim.oghma
sim.json
jv.dat
device.csv
aabb.csv
```

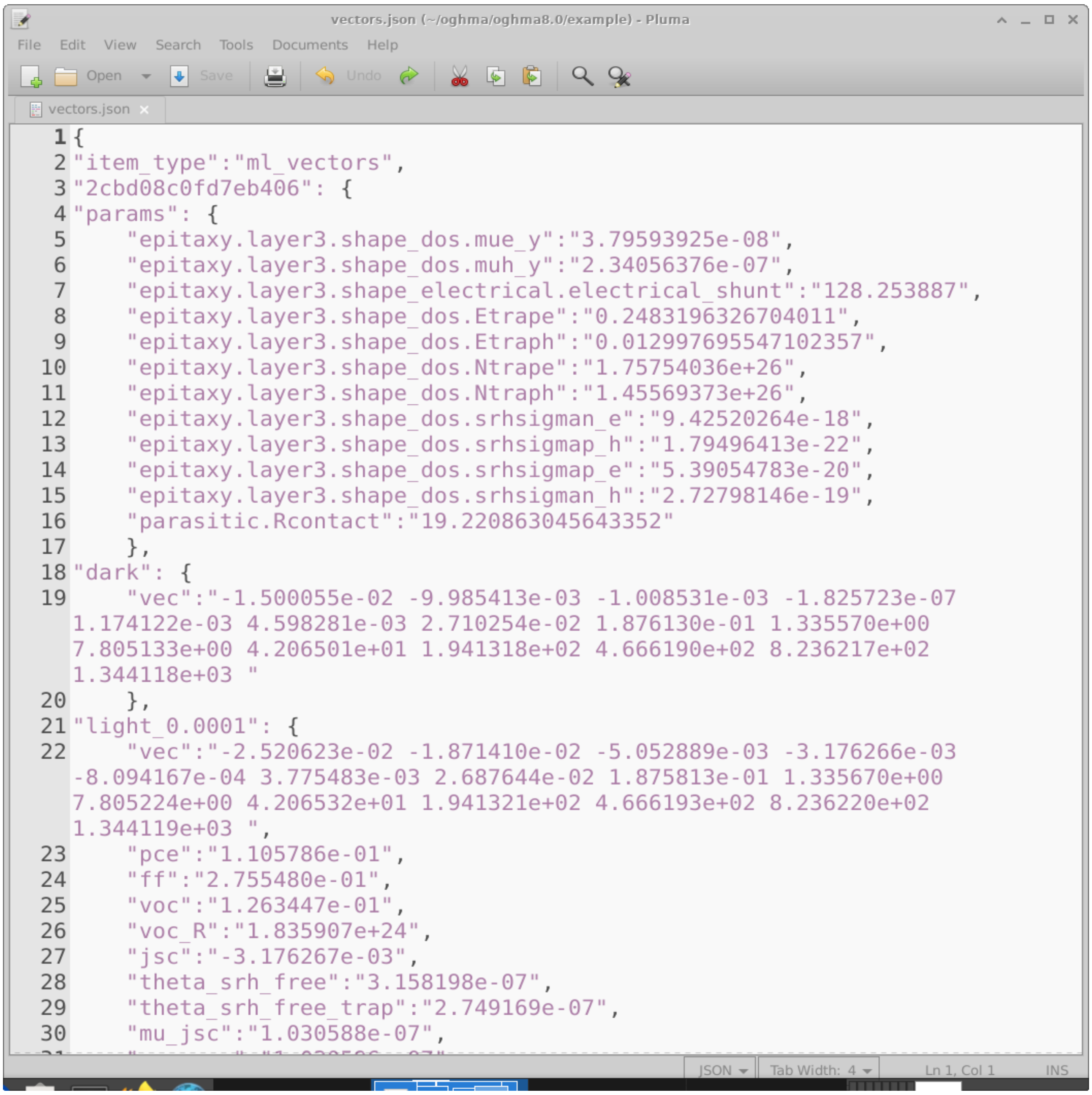
Figure 18.8: Left: The content of the *Example* directory; Right inside archive0.zip

window 18.2, OghamNano will open each virtual simulation and compile the it to a vectors file. The vectors file can be seen in Figure 18.9.

This vectors file is a json file containing all the data needed for training a machine learning algorithm. Each *virtual device* in the data set will have a section in the file. So for example in this figure we are looking at the section relating to device 2cbd08c0fd7eb406 (line 3), we can see under the heading *params* what electrical parameters were randomly chosen for this device. Followed by the vectors representing the dark JV curve (line 19) and the light JV curve at 0.0001 Suns (line 22). These values in this example are in Am^{-2}, however they will always be the same as the file from which they are extracted. So in this case jv.dat contains Volts v.s. Am^{-2} therefore the values of the vector are Am^{-2}. After this key device parameters such as PCE are dumped. There will be one entry in this file for each virtual device generated. Once you have this file, you can feed it into the machine learning algorithm of your choice. Some users report that it is easier to feed this data into tensor flow if it is first converted into a CSV file, this can be done using the standard Python libraries.

18.1.3 Setting up machine learning algorithms to process the data

The above section describes how to generate machine learning data sets. Setting up the algorithm it's self is a large topic on it's own. However there are many good examples at https://www.tensorflow.org/.


vectors.json (~/oghma/oghma8.0/example) - Pluma
File Edit View Search Tools Documents Help
Open Save Undo
vectors.json
JSON Tab Width: 4 Ln 1, Col 1 INS

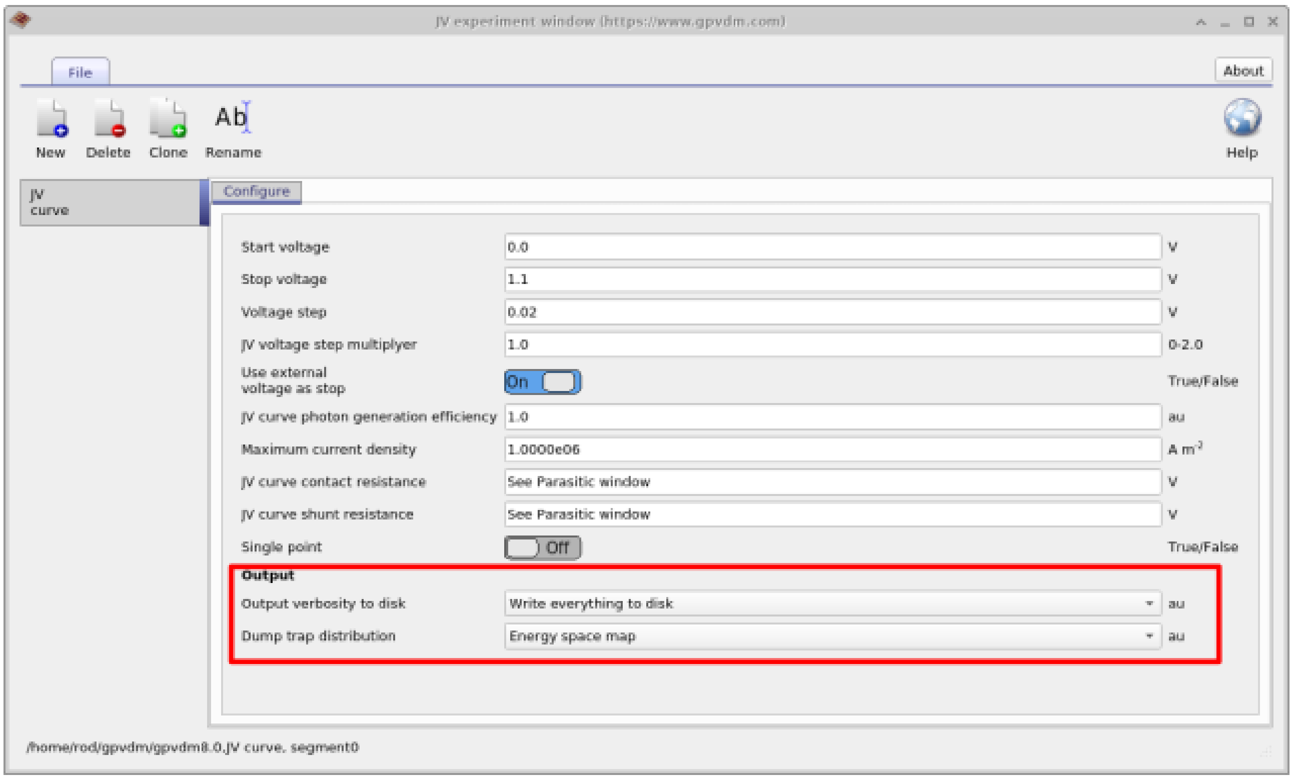
```
{
"item_type":"ml_vectors",
"2cbd08c0fd7eb406": {
"params": {
    "epitaxy.layer3.shape_dos.mue_y":"3.79593925e-08",
    "epitaxy.layer3.shape_dos.muh_y":"2.34056376e-07",
    "epitaxy.layer3.shape_electrical.electrical_shunt":"128.253887",
    "epitaxy.layer3.shape_dos.Etrape":"0.2483196326704011",
    "epitaxy.layer3.shape_dos.Etraph":"0.012997695547102357",
    "epitaxy.layer3.shape_dos.Ntrape":"1.75754036e+26",
    "epitaxy.layer3.shape_dos.Ntraph":"1.45569373e+26",
    "epitaxy.layer3.shape_dos.srhsigman_e":"9.42520264e-18",
    "epitaxy.layer3.shape_dos.srhsigmap_h":"1.79496413e-22",
    "epitaxy.layer3.shape_dos.srhsigmap_e":"5.39054783e-20",
    "epitaxy.layer3.shape_dos.srhsigman_h":"2.72798146e-19",
    "parasitic.Rcontact":"19.220863045643352"
    },
"dark": {
    "vec":"-1.500055e-02 -9.985413e-03 -1.008531e-03 -1.825723e-07 1.174122e-03 4.598281e-03 2.710254e-02 1.876130e-01 1.335570e+00 7.805133e+00 4.206501e+01 1.941318e+02 4.666190e+02 8.236217e+02 1.344118e+03 "
    },
"light_0.0001": {
    "vec":"-2.520623e-02 -1.871410e-02 -5.052889e-03 -3.176266e-03 -8.094167e-04 3.775483e-03 2.687644e-02 1.875813e-01 1.335670e+00 7.805224e+00 4.206532e+01 1.941321e+02 4.666193e+02 8.236220e+02 1.344119e+03 ",
    "pce":"1.105786e-01",
    "ff":"2.755480e-01",
    "voc":"1.263447e-01",
    "voc_R":"1.835907e+24",
    "jsc":"-3.176267e-03",
    "theta_srh_free":"3.158198e-07",
    "theta_srh_free_trap":"2.749169e-07",
    "mu_jsc":"1.030588e-07",
```

Figure 18.9: The final vectors file in json format. This file can be fed into machine learning model as a training set.

Chapter 19

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 19.1



Start voltage	0.0	V
Stop voltage	1.1	V
Voltage step	0.02	V
JV voltage step multiplyer	1.0	0-2.0
Use external voltage as stop	On	True/False
JV curve photon generation efficiency	1.0	au
Maximum current density	1.0000e06	A m^{-2}
JV curve contact resistance	See Parasitic window	V
JV curve shunt resistance	See Parasitic window	V
Single point	Off	True/False
Output		
Output verbosity to disk	Write everything to disk	au
Dump trap distribution	Energy space map	au

/home/rod/gpvdm/gpvdm8.0,JV curve, segment0

[H]

Figure 19.1: 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 19.1). 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 19.2.

19.1 Snapshots directory - dir

The snapshots directory (see figure 19.2) allows the user to plot all internal solver parameters. For example figure 19.3 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.

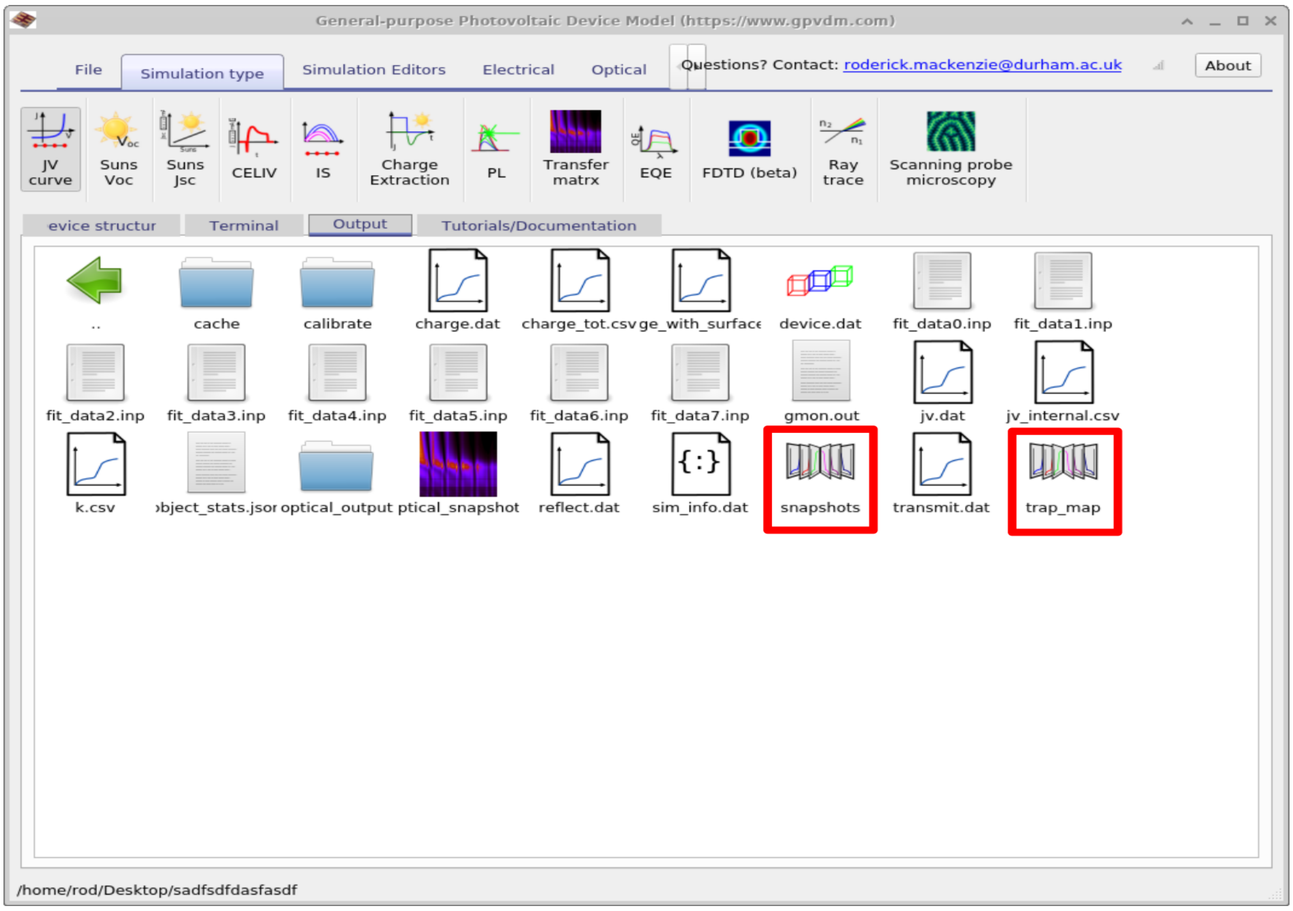
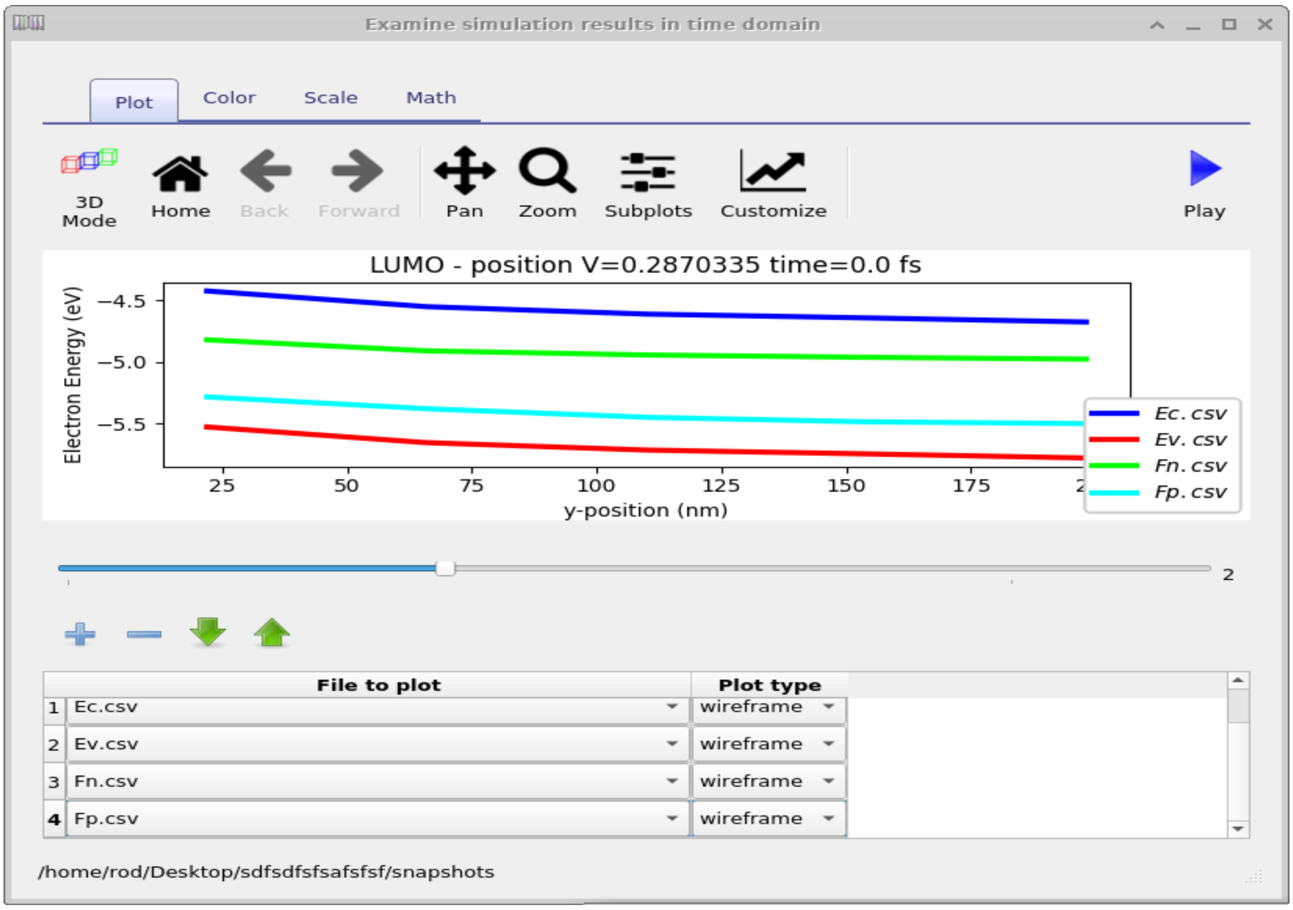


Figure 19.2: The file viewer showing the snapshots and trap map directory



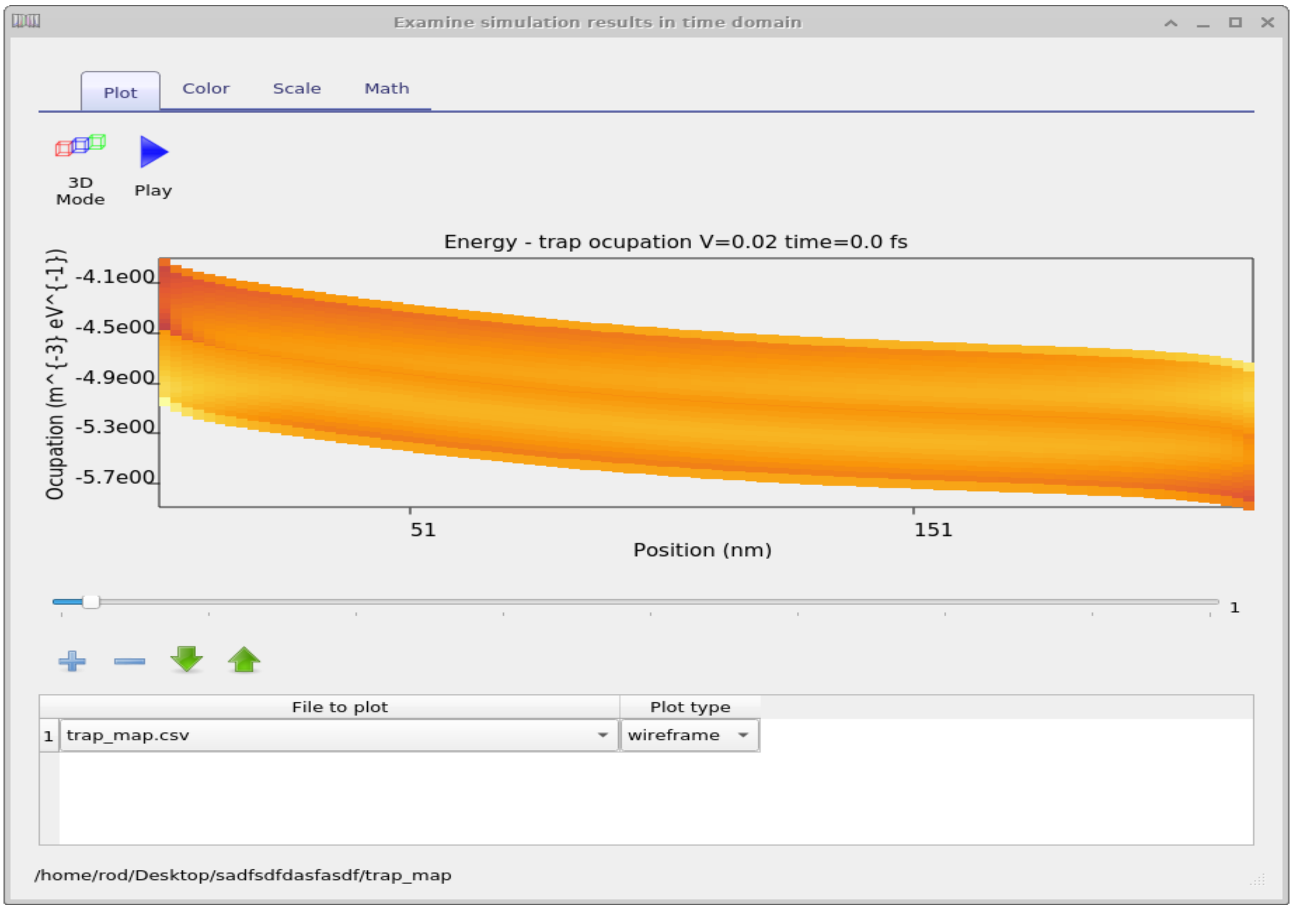
	File to plot	Plot type
1	Ec.csv	wireframe
2	Ev.csv	wireframe
3	Fn.csv	wireframe
4	Fp.csv	wireframe

/home/rod/Desktop/sdfsdfsfsafsfsf/snapshots

Figure 19.3: Using the snapshots tool to view the conduction band, valance band and quasi Fermi-levels

19.2 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 19.4


Examine simulation results in time domain
Plot
Color
Scale
Math
3D Mode
Play
Energy - trap ocupation V=0.02 time=0.0 fs
Ocupation (m^{-3} eV^{-1})
-4.1e00
-4.5e00
-4.9e00
-5.3e00
-5.7e00
51
151
Position (nm)
1

	File to plot	Plot type
1	trap_map.csv	wireframe

/home/rod/Desktop/sadfsdfdasfasdf/trap_map

Figure 19.4: Plotting the position and energy dependence of carriers using the trap map tool

19.3 Optical snapshots - dir

Contains results of the optical simulations.

19.4 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.

19.5 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.[18]

19.5.1 Optical simulation

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

19.6 File formats

Almost all input and output files associated with OghmaNanoare 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.

19.6.1 .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.. OghmaNanois moving from .dat files to .csv files.

19.6.2 .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.

19.6.3 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 OghmaNanowill dump the same json header as used in the csv file but then dump a series of C floats representing the data.

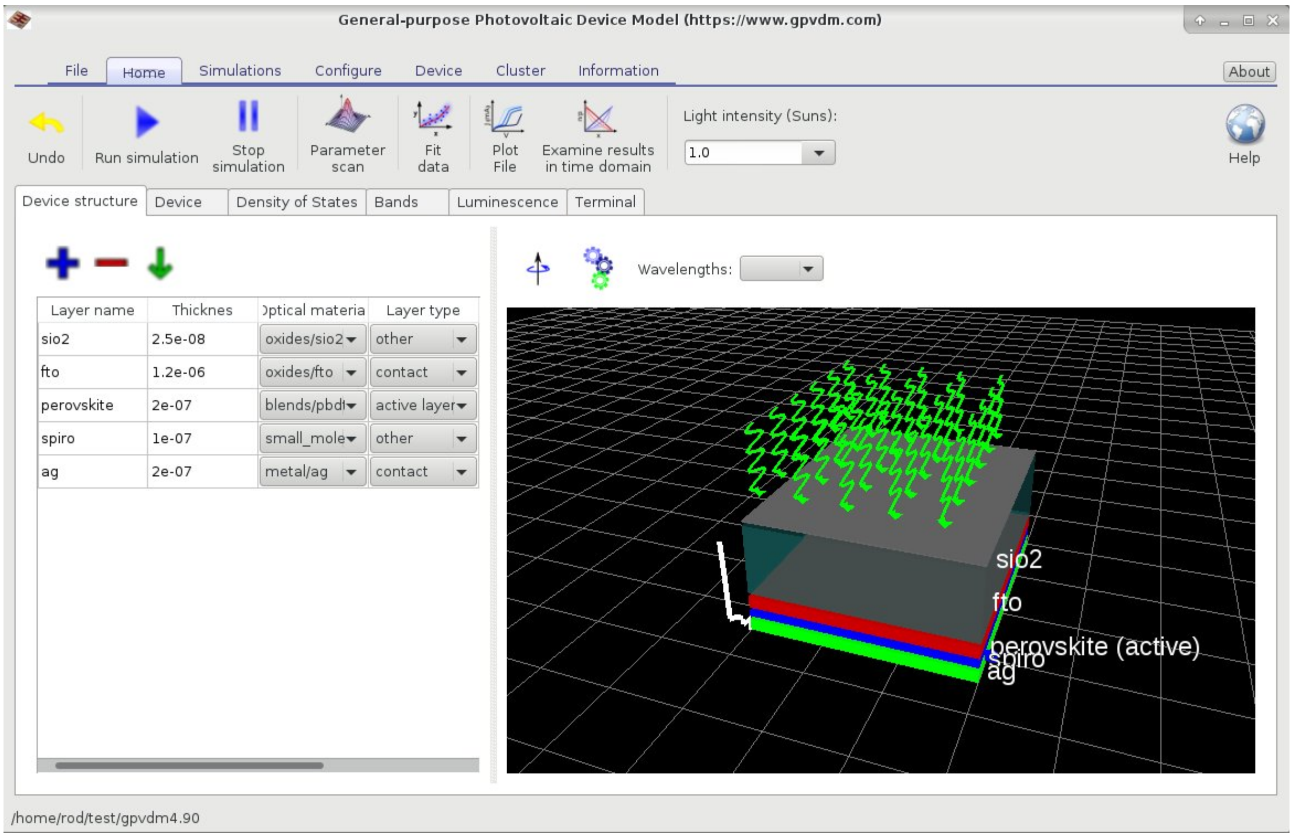
Chapter 20

Troubleshooting

20.1 Windows gives warms me the software is unsigned

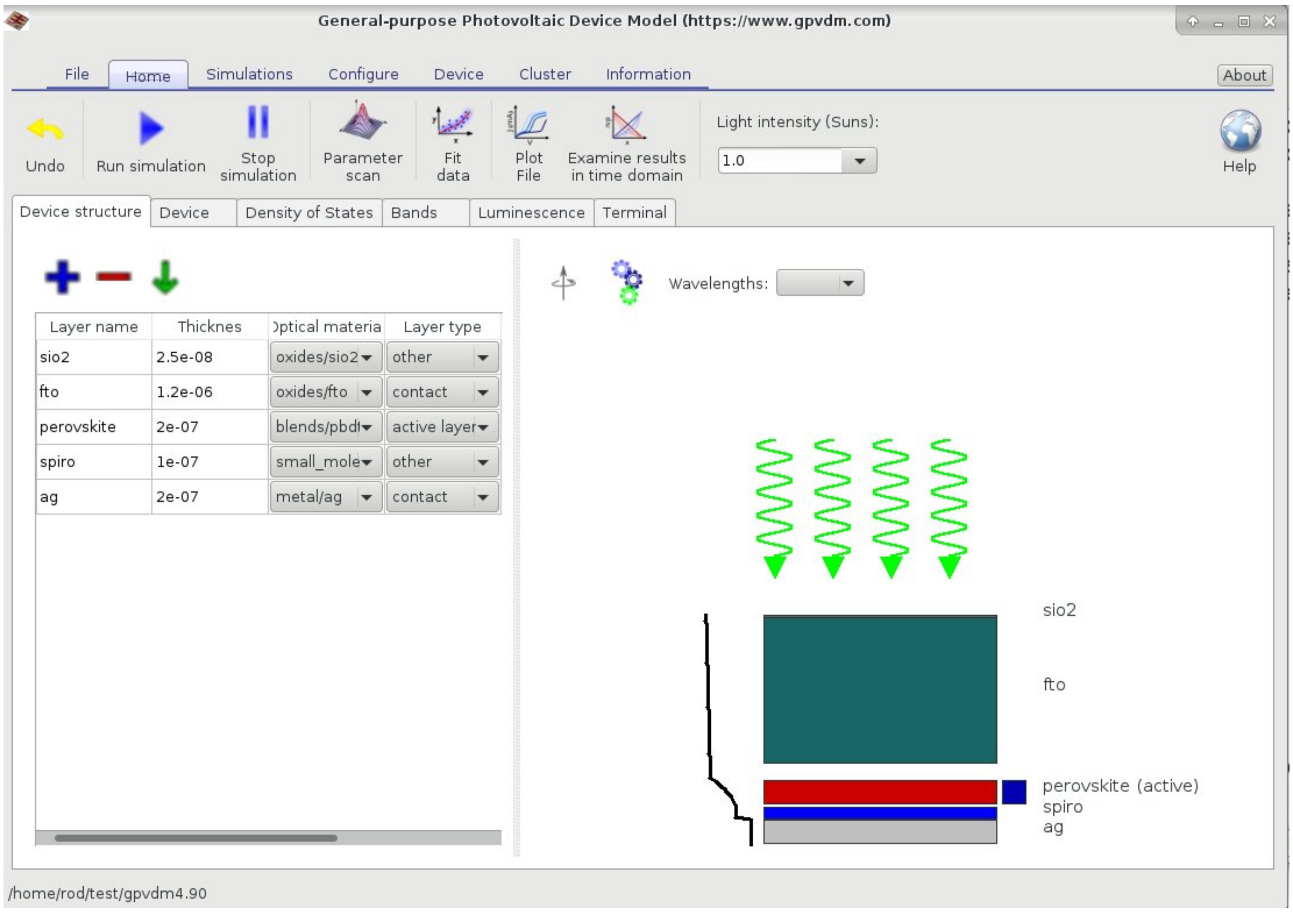
20.2 Why don't I get a 3D view of the device

If your simulation window looks like figure 20.2 and not like figure 20.1. 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.



Layer name	Thicknes	Optical materia	Layer type
sio2	2.5e-08	oxides/sio2	other
fto	1.2e-06	oxides/fto	contact
perovskite	2e-07	blends/pbdt	active layer
spiro	1e-07	small_mole	other
ag	2e-07	metal/ag	contact

Figure 20.1: OghmaNanowith working 3D acceleration hardware.



Light intensity (Suns): 1.0

Layer name	Thicknes	Optical materia	Layer type
sio2	2.5e-08	oxides/sio2	other
fto	1.2e-06	oxides/fto	contact
perovskite	2e-07	blends/pbdt	active layer
spiro	1e-07	small_mole	other
ag	2e-07	metal/ag	contact

Figure 20.2: OghmaNanowith no 3D acceleration hardware.

Chapter 21

FAQ

21.1 Section

21.1.1 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 efficienceis 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.

21.1.2 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.

21.2 Excited states

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

$$\frac{dN_S}{dt} = \frac{1}{4}R_{free} + \frac{1}{4}\kappa_{TT}N_T^2 - (\kappa_S + \kappa_{ISC})N_s - (\frac{7}{4}\kappa_{SS}N_S - \kappa_{ST}N_T)N_S \tag{21.1}$$

$$\frac{dN_t}{dt} = \frac{3}{4}R_{free} + \kappa_{ISC}N_S + \frac{3}{4}\kappa_{SS}N_S^2 \tag{21.2}$$

$$\frac{dN_{SD}}{dt} = \frac{\kappa_{FRET}}{N_{DOP}}(N_{DOP} - N_{SD} - N_{TD})N_s + \frac{1}{4}\kappa_{TTD}N_{TD}^2 - (\frac{7}{4}\kappa_{SSD}N_{SD} + \kappa_{STD}N_{TD})N_{SD} \tag{21.3}$$

Chapter 22

Legal

22.1 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.

22.2 Copyright of the manual

This manual is released under CC-BY license.

22.3 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 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. Periodicity 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. Chabinyc, 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. Chabinyc, 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] Photolitherland. *From Wikipedia, see html link in bib file for full info.*

[13] B. Mills. *From Wikipedia, see html link in bib file for full info.*

[14] I. Vighetto. *From Wikipedia, see html link in bib file for full info.*

[15] R. C. MacKenzie, T. Kirchartz, G. F. Dibb, and J. Nelson. *The Journal of Physical Chemistry C*, 2011, 115 19 9806–9813.

[16] E. M. Azoff. *Solid State Electronics*, September 1987, 30 9 913–917.

[17] F. NÉRY and J. Matos. *Proceedings of the 14th European Colloquium on Theoretical and Quantitative Geography*, page 23.

[18] T. Zhan, X. Shi, Y. Dai, X. Liu, and J. Zi. *Journal of Physics: Condensed Matter*, 2013, 25 21 215301.

[19] P. Peumans, A. Yakimov, and S. R. Forrest. *Journal of Applied Physics*, 2003, 93 7 3693–3723.

[20] J. Schneider. *Understanding the Finite-Difference Time-Domain Method*. 2010.

[21] S. Solak, S. Shishegaran, A. C. Hübler, and R. C. MacKenzie. *Solar RRL*, 2021, 5 12 2100787.

Index

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 may other models OghmaNanois 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, OghmaNanohas become one of they key device modelling tools used by the scientific community developing the next generation of opto-electronic devices.

This book written by the author of OghmaNanoexplains 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..