PVMOS manual

B.E. Pieters*
Institut für Energie und Klimaforschung - IEK5 Photovoltaik,
Forschungszentrum Jülich, 52425 Jülich, Germany
(Dated: March 31, 2015)

I. INTRODUCTION

This is (or rather will be as the current state of this document is far from finished) the manual for the Photo-Voltaic MOdule Simulator (PVMOS). PVMOS is an ordinary differential equation solver using finite-differences specifically designed to electrically model solar modules. For more information on how that works I refer the reader to [1]. The purpose of this document is to document how PVMOS is operated and installed. In the following sections I discuss in order, the installation, basic usage and operating principles and finally a detailed discussion of all available functions.

Before we continue, here are some legalities:

```
DISCLAMER:
PVMOS Copyright (C) 2014 B. E. Pieters
This program comes with ABSOLUTELY NO WARRANTY. This is free software, and you are welcome to
redistribute under certain conditions.  You should have received a copy of the GNU General
Public License along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

In order to stress the ABSOLUTELY NO WARRANTY bit, here a quote from R. Freund:

```
For all intent and purpose, any description of what the codes are doing should be construed as
being a note of what we thought the codes did on our machine on a particular Tuesday of last
year.  If you're really lucky, they might do the same for you someday.  Then again, do you really
feel *that* lucky?
```

II. BASIC USAGE AND OPERATING PRINCIPLES

The input for PVMOS is a plain text file with commands. To solve a problem PVMOS is typically called from the command line with as an argument the filename describing the problem. With these input files you can specify the geometry of your solar cell/module including the local properties such as electrode sheet resistances and solar cell properties. You can also specify which calculations you want PVMOS should perform and what data to save. A call to PVMOS from the command line looks like this

```
pvmos [verbose-option] <input-file>
```

where `[verbose-option]` is an option which how much information PVMOS outputs to stdout, and `<input-file>` is the plain text input file. We first describe the mesh data structure in more detail in the next section.

In its essence PVMOS is a Poisson solver. The Poisson equation is solved for several stacked, 2D "electrodes", where each electrode is coupled to the electrodes above and below (note that a stack of 2D electrodes makes a 3D structure). The electrodes itself simply conductors and behave linearly (Ohmic). However, the connection between the electrodes can be non-linear (e.g. a diode). Now ther are several limitations of the structures that PVMOS handles. The main limitation is that the meshes are 2D as in PVMOS all electrodes share the same 2D mesh. This means that PVMOS is specifically designed for flat layered structures, a less than optimal performance is to be expected for different that flat layered geometries. PVMOS uses straight forward finite-differences to solve the coupled Poisson equations. As such the 2D meshes in PVMOS divide the 2D surface in rectangular elements. Now every element in the mesh must be associated with certain properties. Storing properties on a per element basis woul put a heavy

*Electronic address: b.pieters@fz-juelich.de

burdon on the memory resources. For that reason elements are grouped into "area's" where each ares constitures a certain combination of properties. A mesh therfore also contains a list of all area's and every element is a member of one of the area's in the mesh (note that an element is always assigned to one area, i.e. you need at least one area for every unique combination of local properties).

In PVMOS we can define more than one mesh at the same time (this is useful as you can build meshes from several meshes, e.g. join two meshes for single cells to one mesh with two series connected cells). In order to reference one particular mesh, each mesh has a name. To reference an area within a mesh you can refer to `<mesh-name>.<area-name>`, i.e. the name of the mesh followed by a dot and the name of the area within that particular mesh. Sometimes we need to select elements in a mesh (for example when we want to assign a set of elements to a certain area). To this end each mesh has a list of selected elements. If you select elements in a mesh the list is occupied by the element ID's, after which you can do operations on the selected elements. Note that elements are selected on a per mesh basis.

The input file is parsed by PVMOS, which sequentially processes the file. PVMOS provides functions to generate and manipulate meshes such that you can generate meshes describing the problem by a sequence of commands. To this end each mesh you define has a name to reference it by. The following section provides a command-reference.

III. INSTALLATION INSTRUCTIONS

PVMOS is a command-line application written entirely in C. For the operation PVMOS depends on several libraries, most notably `cholmod`, for the solving of sparse linear systems. PVMOS has been tested on Linux and Windows systems. To install PVMOS you need to either compile the source, for which a Makefile is provided, or you need to download precompiled binaries for your system (I only provide pre-compiled versions for windows).

A. Pre-compiled binaries (Windows)

A word of warning: I rarely use windows systems. As such the here provided information about getting PVMOS to work on windows systems is a description of how I got PVMOS to work on one particular system on one particular Wednesday afternoon in December. I make no claims regarding the correctness of the provided information as generally I know very little about what I am doing, certainly if it involes a computer with Microsoft Windows. Please bear that in mind before you accuse me of providing false information of even damaging your system. This text is meant to provide some pointers in the hope it helps, I am in no way responsible for what you do on your system.

There are two pre-compiled versions, 32 and 64 bit. If your system supports it I strongly suggest to use 64 bit as 32bit applications are rather limited in the maximum allocatable memory space. The Pinup example will most likely not run with the 32bit version due to this memory limitation. The binary distribution consists of an executable with several dll's. The installation procedure for PVMOS is rather manual and no installer is provided. So here is the procedure for a binary install:

1. Download the binary
2. Unpack the files in a directory of your choise
3. Optional: Add the path to your PVMOS executable to the PATH variable

1. Setting the "PATH" variable

Here I describe how to edit your path variable in Windows 7. First another word of warning. Adding things to the path variable is mostly harmless, however, if you break your path string (e.g. delete a part) you may seriously impair the functioning of programs that need the path variable. Follow these instructions on your own risk!

1. Click on the start menu
2. type "PATH in the "search programs and files, you should now find options to edit the path variable for the system of for the current user, select to edit it for the current user
3. Select the path variable
4. Click on Edit

5. Scroll to the end of the string and add `;<installation directory>`, where `<installation directory>` points to the location of your PVMOS executable
6. Click on OK
7. Click on OK

If you properly edited your path variable you should be able to call PVMOS from the command prompt, from any directory, just by typing "pvmos", i.e. you do not need to include the path to your PVMOS executable.

2. Installing mkpvmosmesh

The "mkpvmosmesh" library is a dynamic library for GNU Octave which allows one to save PVMOS meshes from within Octave. Its use is demonstrated in the "Pinup" example. Again you can choose between precompiled binaries or compile it yourself. The precompiled version will not work with all versions of Octave, thus to use the precompiled version you need to install the correct version of Octave. The version of Octave I used is provided by "octave-forge". Octave-forge provides several version of Octave. I used the version 3.6.4 of Octave that was compiled with the MinGW compiler suite version gcc4.6.2 (Octave3.6.3_gcc4.6.2). The provided libryry will not work with the version of Octave that is compiled with Visual Studio. The precompiled library is 32bit as the used version of Octave is too. As of this writing no pre-compiled 64bit version of Octave is available (at least not compiled with the MinGW suite). For this reason there is no 64 bit version of mkpvmosmesh.

Two notes on using GNU Octave under windows.

1. You can add Octave to your path variable, that way you can easily call octave from anywhere on the commandline.
2. you may find that on windows Octave always starts in the same directory, this is to me unexpected and undesired. I got Octave to start there where I call it (so I can execute an octave script by typing `octave <path to octave script>`. To this end I edited the file `<Octave-installation-dir>\share\site\m\startup\octaverc` by commenting out the line

   ```
   cd (getenv('USERPROFILE'))
   ```

B. Compiling from source

1. Install PVMOS's dependencies (`BLAS` and `CHOLMOD`). These libraries should be readily available in most linux distributions. Take care to install the development files (i.e. headers) for these libraries too
2. Optionally edit the Makefile, in particular you can change the BLAS library to link to (which may be relevant for performance, see below) or change the install directory
3. type "make", and "make install"
4. Optionally type "make mkpvmosmesh" to compile the mkpvmosmesh library for GNU Octave (this requires a GNU Octave install with a working mkoctfile command).

The performance of PVMOS is typically strongly dependent on the performance of the sparse linear solver (i.e. `cholmod`). For an optimal performance an optimized `BLAS` library must be used (the reference `BLAS` is comparatively slow). For an optimized `BLAS` there are several options. One option is to use ATLAS which is available for all common CPU architectures and gives a very decent performance. On some architectures you can use `OpenBLAS`, which gives a very good performance (`OpenBLAS` is an actively developed fork of the now unmaintained `GoTo BLAS`). Some CPU manufacturers also publish their own optimized `BLAS` libraries for their CPU's (e.g. Intel and AMD). Per default the makefiles are set up to use `OpenBLAS` as it provides a good performance and is freely available.

IV. GETTING STARTED

In this section we give some pointers to get you started.

A. Recommended software

We recommend the use of several pieces of software as we use those ourselves and as such PVMOS was partly structures to be compatible with specific tools. Most notable of this is Gnuplot. Some of PVMOS output formats are specifically designed to be compatable with Gnuplot. Specifically the format used to make map plots of data on the irregular meshes, I suspect other plotting tools do not work well with this output format (I do not know for sure as I do not use other tools for plotting). The use of Gnuplot is recommended especially for inspecting the mesh definitions to see if the defined geometry is correct (the output of the `printarea` command, combined with the `printpar` command).

If you want to define complex geometries there are basically three ways:

1. Use the PVMOS scripting language. The advantages of this method is that it can be very accurate and fast and is independent of any other software, the downside is that it quickly becomes tedious
2. Use a set of bitmap "masks" in combination with GNU octave and the mkpvmosmesh library for GNU octave. This is a powerful way to define complicated geometries. For this you need to set up and install GNU Octave and the mkpvmos library. For high resolutions memory usage and performance may become an issue.
3. Define geometries using polygons. PVMOS has good support for polygons to select and specify various areas in your device. Complicated shapes may be created in a graphics program such as inkscape and exported to PVMOS. For this I have an extension to export data from inkscape for use in PVMOS. In combination with the trace bitmap feature of inkscape you can quickly define appropriate polygons for complicated geometries if you provide a bitmap image, e.g. you can create a polygon for the metalization pattern of a device by tracing a picture. This method potentially has a lower memory footprint compared to the GNU Octave option.

B. Gnuplot

The data-format that PVMOS uses for element-wise data export is laid out such that Gnuplot can make sense out of the data. Other plotting tools may not support this format. The element-wise data export allows you to export data from the mesh on a per-element basis and is particularly useful for inspecting meshes to ensure a correct mesh definition. For this reason I recommend the use of Gnuplot. Other data formats thet PVMOS should not be a problem for other plotting tools.

C. GNU Octave

This is only relevant if you want to define meshes using bitmap masks.

D. Inkscape

Inkscape is a powerfull vector graphics tool. One of the graphics objects Inkscape supports are "paths". In its simplest form a path is simply a polygon. As PVMOS can use polygons when defining a mesh it makes sense to use a tool such as Inkscape to create the polygons as Inkscape provides a powerfull, graphical, means to do so. In order to get the polygon data out of inkscape I created a simple extension which extracts the data out of a polygon. To use this extention select a path with polygon data and select the ExportXY extension, a popup window will come up with coordinates you can select and paste in a polygon data file. Note however that Inkscape paths may also be bezier curves, which PVMOS does not support. To use the extension is is thus required that the path data is forced to be a polygon. For this you can use the flatten Bezier extention in Inkscape. I have not fully dug into Inkscape's internal units. This can be tricky as Inkscape may use all sorts of transforms on objects. This means that the polygon data may need to be scaled, rotated, mirrored and shifted to get the data in the correct unit. For this I provide a small CLI program "transformpolygon".

V. PVMOS INPUT FILE FORMAT

A. Command Reference

Creating Meshes

Keyword	Description
`newmesh`	Create a new, rectangular mesh. The command takes seven arguments: 1. `x1`, x-coordinate of the lower left corner 2. `y1`, y-coordinate of the lower left corner 3. `x2`, x-coordinate of the upper right corner 4. `y2`, y-coordinate of the upper right corner 5. `Nx`, Number of elements in x direction 6. `Ny`, Number of elements in y direction 7. `mesh-name`, Name of the new mesh
`joinmesh`	Create new mesh by joining two meshes. Make sure the meshes touch but do not overlap. The function takes offset values as input which allow you to "shift" the second mesh to align it to the first. The command takes 5 arguments: 1. `x_off`, x-offset in coordinate system of the second mesh 2. `y_off`, y-offset in coordinate system of the second mesh 3. `mesh1-name`, Name of the first mesh 4. `mesh2-name`, Name of the second mesh 5. `mesh3-name`, Name of the resulting mesh
`joinmesh_h`	Create new mesh by joining two meshes. Make sure the meshes touch but do not overlap. The function takes a y-offset value as input which allow you to "shift" the second mesh in the y-direction to align it to the first. The x-offset value is the maximal x-value found in the first mesh. The command takes 4 arguments: 1. `y_off`, y-offset in coordinate system of the second mesh 2. `mesh1-name`, Name of the first mesh 3. `mesh2-name`, Name of the second mesh 4. `mesh3-name`, Name of the resulting mesh
`joinmesh_v`	Create new mesh by joining two meshes. Make sure the meshes touch but do not overlap. The function takes an x-offset value as input which allow you to "shift" the second mesh in the x-direction to align it to the first. The y-offset value is the maximal y-value found in the first mesh. The command takes 4 arguments: 1. `x_off`, x-offset in coordinate system of the second mesh 2. `mesh1-name`, Name of the first mesh 3. `mesh2-name`, Name of the second mesh 4. `mesh3-name`, Name of the resulting mesh
`dupmesh`	Duplicate a mesh. The command takes 2 arguments: 1. `mesh1-name`, Name of the mesh to be duplicated 2. `mesh2-name`, Name of the resulting copy
`add_electrode`	Adds an electrode to a certain mesh. The command takes one argument: 1. `mesh-name`, Name of the mesh

Transforming Meshes

Keyword	Description
`move`	Move a mesh in x- and y-directions. The command takes three arguments: 1. `mesh-name`, Name of the mesh 2. `delta x`, move distance in x-direction 3. `delta y`, move distance in y-direction

`rotate` Rotate mesh around a center coordinate. Rotation can only be performed over multiples of 90 degrees. The command takes four arguments:

1. `mesh-name`, Name of the mesh
2. `x`, x-coordinate of rotation center
3. `y`,y-coordinate of rotation center
4. `d`, degrees to rotate over (must be a multiple of 90)

`flipx` Mirror a mesh in the y-axis. The command takes one argument:

1. `mesh-name`, Name of the mesh

`flipy` Mirror a mesh in the x-axis. The command takes one argument:

1. `mesh-name`, Name of the mesh

`scale` Scale all coordinates in a mesh. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `f`, scaling factor

`scalex` Scale all x-coordinates in a mesh. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `f`, scaling factor

`scaley` Scale all y-coordinates in a mesh. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `f`, scaling factor

`boundingbox` Change the mesh bounding box. This argument scales and moves a mesh such that it fits in a specified rectangle (the bounding box). The mesh is either scaled preserving the aspect ratio or without preserving it. The command takes six arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the bounding box
3. `y1`, y-coordinate of the lower left corner of the bounding box
4. `x2`, x-coordinate of the upper right corner of the bounding box
5. `y2`, y-coordinate of the upper right corner of the bounding box
6. `R`, Argument can be either 0, do not preserve mesh aspect ratio, or 1, preserve mesh aspect ratio.

Selecting Elements

Keyword	Description
`select_rect`	Select a rectangular area in a mesh. If currently nodes are already selected in the mesh, the command selects the subset of selected nodes within the rectangle. If the latter is not desired use deselect first. The command takes five arguments: 1. `x1`, x-coordinate of the lower left corner of the selected rectangle 2. `y1`, y-coordinate of the lower left corner of the selected rectangle 3. `x2`, x-coordinate of the upper right corner of the selected rectangle 4. `y2`, y-coordinate of the upper right corner of the selected rectangle 5. `mesh-name`, Name of the mesh

`select_rect_contour` Select the contour of a rectangle in a mesh. If currently nodes are already selected in the mesh, the command selects the subset of selected nodes within the rectangle. If the latter is not desired use deselect first. The command takes six arguments:

1. `x1`, x-coordinate of the lower left corner of the selected rectangle
2. `y1`, y-coordinate of the lower left corner of the selected rectangle
3. `x2`, x-coordinate of the upper right corner of the selected rectangle
4. `y2`, y-coordinate of the upper right corner of the selected rectangle
5. `d`, distance from the contour within which elements are selected
6. `mesh-name`, Name of the mesh

`select_circ` Select a circular area in a mesh. If currently nodes are already selected in the mesh, the command selects the subset of selected nodes within the circle. If the latter is not desired use deselect first. The command takes four arguments:

1. `x_c`, center x-coordinate of the selected circle
2. `y_c`, center y-coordinate of the selected circle
3. `r`, radius of the selected circle
4. `mesh-name`, Name of the mesh

`select_circ_contour` Select the contour of a circle in a mesh. If currently nodes are already selected in the mesh, the command selects the subset of selected nodes within the circle. If the latter is not desired use deselect first. The command takes five arguments:

1. `x_c`, center x-coordinate of the selected circle
2. `y_c`, center y-coordinate of the selected circle
3. `r`, radius of the selected circle
4. `d`, distance from the contour within which elements are selected
5. `mesh-name`, Name of the mesh

`select_poly` Select an area within a polygon-contour. In order to use this command you must first load or define a polygon from file with the `load_poly` or `define_poly` commands. If currently elements are already selected in the mesh, the command selects the subset of selected elements within the polygon. If the latter is not desired use deselect first. If a polygon circles an element more than once the node is selected if the node is circled an uneven number of times. This allows one to define polygon areas with holes. To make a hole, define one outer part of the polygon and one inner part and have a break between these tho parts, i.e. the two parts should not be connected by a line segment, (see `load_poly` and `loaddefine_poly`). The command takes one argument.

1. `mesh-name`, Name of the mesh

`select_poly_contour` Select an area around a polygon-contour. In order to use this command you must first load or define a polygon from file with the `load_poly` or `define_poly` commands. If currently elements are already selected in the mesh, the command selects the subset of selected elements near to the polygon. If the latter is not desired use deselect first. This command is often useful to refine the mesh along the polygon before using `select_poly` to assign elements to a new area. It may also be useful to define things such as cracks. If no line segment is present between two seubsequent coordinates in the polygon (i.e. a break in the polygon), no nodes are selected between these coordinates. The command takes three arguments.

1. `distance`, Distance from the polygon
2. `loop`, Argument takes either 0 (not looped) or 1 (looped). In looped modus the last point in the polygon is connected to the first point.
3. `mesh-name`, Name of the mesh

`load_poly` Load a polygon from file. This command is used in conjunction with the `select_poly` and the `select_poly_contour` commands. Once a polygon is loaded you can use it to select until a new `load_poly` command is given. Polygons may contain breaks (i.e. absence of a line segment between two coordinates). The command takes one argument.

1. `file-name`, Name of the file describing the polygon (one 2D coordinate per line, i.e., two columns, 1: x-coordinate, 2: y-coordinate, empty lines indicate a break)

`define_poly` Define polygon within the input file. See also the `load_poly` command for an alternative method. The difference between this command and the `load_poly` command is that with this command you can define the polygon within the PVMOS input file. As such it is best suited for simple polygons. The `define_poly` command marks the start and the end of a table defining the polygon. Between the two `define_poly` commands, one coordinate (x- and y-value) per line is expected and an empty line indicates a break, i.e.,

`define_poly`
x_1 y_1
x_2 y_2
x_3 y_3
... ...
x_N y_N

x_{N+1} y_{N+1}
... ...
x_M y_M
`define_poly`

where (x_i, y_i) is the i-th coordinate of the polygon and between coordinates N and $N+1$ there is a break.

`select_area` Select all nodes assigned to a given area. If currently nodes are already selected in the mesh, the command selects the subset of selected nodes which are assigned to the given area. If the latter is not desired use deselect first. The command takes one argument.

1. `area-name`, Name of the mesh and area (<mesh-name>.<area-name>)

`deselect` deselects a selection within a mesh. The command takes one argument.

1. `mesh-name`, Name of the mesh

Manually changing the mesh topology

Keyword	Description
`split_x`	Split selected elements in x-direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_y`	Split selected elements in y-direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_xy`	Split selected elements in both x- and y-direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh
`split_long`	Split selected elements in thier longest direction. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes one argument 1. `mesh-name`, Name of the mesh

`split_coarse` Split selected elements until the node-edges are all smaller than a given length. If no elements are selected, all elements are split. As the topology of the mesh changed all selected nodes in the mesh are un-selected after this command. The command takes two arguments

1. `mesh-name`, Name of the mesh
2. `l`, Maximum edge length

`resolve_rect` Split elements along the edges of a rectangle untill all element edges are smaller than a given length. This command can be used to ensure a particular rectangle fits accurately in the mesh. The command takes six arguments

1. `mesh-name`, Name of the mesh
2. `l`, Maximum edge length
3. `x1`, x-coordinate of the lower left corner of the selected rectangle
4. `y1`, y-coordinate of the lower left corner of the selected rectangle
5. `x2`, x-coordinate of the upper right corner of the selected rectangle
6. `y2`, y-coordinate of the upper right corner of the selected rectangle

`resolve_circ` Split elements along the circumference of a circle untill all element edges are smaller than a given length. This command can be used to ensure a particular circle fits accurately in the mesh. The command takes five arguments

1. `mesh-name`, Name of the mesh
2. `l`, Maximum edge length
3. `x_c`, center x-coordinate of the selected circle
4. `y_c`, center y-coordinate of the selected circle
5. `r`, radius of the selected circle

`resolve_poly` Split elements along the contour of a polygon untill all element edges are smaller than a given length. This command can be used to ensure a particular polygon fits accurately in the mesh. The command requires a polygon to be defined with either `define_poly` or `load_poly`. Breaks in the polygon are not resolved, i.e. the routine only resolves the line segments within the polygon. The command takes two arguments

1. `mesh-name`, Name of the mesh
2. `l`, Maximum edge length

`simplify` Attempt to simplify a mesh. If elements are selected they are un-selected as the topology of the mesh changed. The command takes one argument

1. `mesh-name`, Name of the mesh

Saving and loading meshes

Keyword	Description
`savemesh`	Save a mesh to file in the PVMOS binary format, so it can be loaded again at a later time (see the `loadmesh` command). The command takes two arguments.

1. `mesh-name`, Name of themesh to be saved.
2. `file-name`, filename to save the mesh to.

`loadmesh` Load a mesh saved to file in the PVMOS binary format (see the `savemesh` command). The command takes two arguments.

1. `file-name`, filename of the file containing the mesh data.
2. `mesh-name`, Name to assign to the loaded mesh

Element-wise export of data

Keyword	Description

`printmesh` Export the mesh in a manner that is plottable with the gnuplot program (www.gnuplot.info/). The resulting plot draws the contour of each element in the mesh. If a selection of elements is made for the mesh, only the selected nodes are plotted. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output file will contain coordinates in columns. For each element the file contains the coordinates of the lower left- and the upper right corners empty line:

```
x1    y1
x2    y1
x2    y2
x1    y2
<empty line>
```

`printconn` Print lateral connections in the electrodes in a format plottable with gnuplot (www.gnuplot.info/). When plotting the file (with vectors) a vector is drawn between the center of each element to the center of the adjacent elements to which it is connected. If a selection of elements is made for the mesh, only the selected nodes are plotted. This routine may be useful when inspecting generated meshes. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output is coordinates in columns. For each element the file contains the following data where `xc`, `yc` is the center of the current element and `xca_i`, `yca_i` is the center coordinate of the i-th adjacent

```
         xc yc xca_1 yca_1
element: xc yc xca_2 yca_2
         ...
```

`printarea` Print the geometry of the mesh which identifies each element and the area it belongs to. The fileformat is laid out such that it is plottable with a surface plot in gnuplot (www.gnuplot.info/). If a selection of elements is made for the mesh, only the selected nodes are plotted. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output file will contain data in columns. The file contains coordinates, the element ID and the corresponding area ID. Note that the parameters for each area can be exported with the `printpars` command. For each element it plots 2 times 2 data lines with an empty line inbetween. Between the data of two elements are two empty lines. This file is formatted such that when plotted with "splot" in gnuplot you can plot a surface for each element in the mesh, which allows you to see the areas in the defined geometry. For each element the folowing data is printed to the file:

```
x1    y1    element-ID area-ID
x1    y2    element-ID area-ID
<empty line>
x2    y2    element-ID area-ID
x2    y1    element-ID area-ID
<empty line>
<empty line>
```

`printV` Print the electrode potentials per element for each stored solution. The output is formatted for gnuplot's splot command, such that a surface plot plots each element individually. If a selection of elements is made for the mesh, only the selected nodes are plotted. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

The output file will contain data in columns. The file contains coordinates followed by the potential in each electrode for each solution. For each element it plots 2 times 2 data lines with an empty line inbetween. Between the data of two elements are two empty lines. This file is formatted such that when plotted with "splot" in gnuplot you can plot a surface for each electrode in each element in the mesh. For each element the folowing data is printed to the file, where the subscripts indicate the electrode index and the superscript the solution index:

`x1` `y1` $V_0^1\ V_1^1\ V_{...}^1\ V_N^1\ V_0^2\ V_1^2\ V_{...}^2\ V_N^2$ ...
`x1` `y2` $V_0^1\ V_1^1\ V_{...}^1\ V_N^1\ V_0^2\ V_1^2\ V_{...}^2\ V_N^2$ ...
`<empty line>`
`x2` `y2` $V_0^1\ V_1^1\ V_{...}^1\ V_N^1\ V_0^2\ V_1^2\ V_{...}^2\ V_N^2$ ...
`x2` `y1` $V_0^1\ V_1^1\ V_{...}^1\ V_N^1\ V_0^2\ V_1^2\ V_{...}^2\ V_N^2$ ...
`<empty line>`
`<empty line>`

`printpar` Print a summary of the parameters per area, including both area-name and area-ID. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

`printIV` Export the IV characteristics of the device. Exports a file with two columns, the first contains the applied voltage and the second the corresponding simulated total current. The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

`printInIp` Integrate external currents over selected elements. Exports a file with four columns, the applied voltage, the integrated current injected from the positive node, the integrated current emitted to the negative node, and the total current over the whole device. This may be used for sanity checking or, if your device has mre than one connection to the applied bias, to distingush between the connections (e.g. extract the current injected through the middle busbar). The command takes two arguments:

1. `mesh-name`, Name of the mesh
2. `file-name`, filename to save the data in.

`surfVplot` Export the electrode voltages for a specific solution. Unlike the `print`-commands like `printV` the data is interpollated and mapped on a regular mesh. The command takes eight arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `Nx`, Number of points in the regular mesh along the x-direction
7. `Ny`, Number of points in the regular mesh along the y-direction
8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken
9. `file-name`, filename to save the data in.

`surfVjplot` Export the junction voltages for a specific solution. Just like in the `surfVplot` command the data is interpollated and mapped on a regular mesh. The junction voltage is defined for the one and two diode models. For other models the junction voltage may not be available, in which case it is set to 0. The command takes eight arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `Nx`, Number of points in the regular mesh along the x-direction
7. `Ny`, Number of points in the regular mesh along the y-direction
8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken
9. `file-name`, filename to save the data in.

`surfPplot` Export the local power density for a specific solution. Just like in the `surfVplot` command the data is interpollated and mapped on a regular mesh. The command takes eight arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `Nx`, Number of points in the regular mesh along the x-direction
7. `Ny`, Number of points in the regular mesh along the y-direction
8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken
9. `file-name`, filename to save the data in.

`surfJplot` Export the current densities in the electrodes and through the solar cells. Unlike the `print`-commands like `printV` the data is mapped on a regular mesh. The command takes eight arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `Nx`, Number of points in the regular mesh along the x-direction
7. `Ny`, Number of points in the regular mesh along the y-direction
8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken
9. `file-name`, filename to save the data in.

`surfEplot` Export the local electric field in x- and y- direction in the electrodes. Just like in the `surfVplot` command the data is mapped on a regular mesh. The command takes eight arguments:

1. `mesh-name`, Name of the mesh
2. `x1`, x-coordinate of the lower left corner of the selected rectangle
3. `y1`, y-coordinate of the lower left corner of the selected rectangle
4. `x2`, x-coordinate of the upper right corner of the selected rectangle
5. `y2`, y-coordinate of the upper right corner of the selected rectangle
6. `Nx`, Number of points in the regular mesh along the x-direction
7. `Ny`, Number of points in the regular mesh along the y-direction
8. `Va`, Applied voltage (if the sepcified voltage is not available the closest value will be taken
9. `file-name`, filename to save the data in.

Manipulating local properties

Keyword	Description
`assign_properties`	Assign nodes to a defined area. If no nodes are selected all nodes in the mesh are assigned to the specified area. The command takes one arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>)
`set_Rel`	Set an electrode resistance per area. If the specified area does not exist it will be newly created. The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `electrode-index`, Index of the electrode. The first electrode has index 0 3. `value`, Sheet resistance value (Ω)
`set_Rvp`	Set the contact resistance between the positive node and an electrode per area. If the specified area does not exist it will be newly created. Together with `set_Rvn` this command allows the application of an extranal voltage. The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `electrode-index`, Index of the electrode. The first electrode has index 0 3. `value`, Contact resistance (Ωcm^2)
`set_Rvn`	Set the contact resistance between the negative node and an electrode per area. If the specified area does not exist it will be newly created. Together with `set_Rvp` this command allows the application of an extranal voltage.The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `electrode-index`, Index of the electrode. The first electrode has index 0 3. `value`, Contact resistance (Ωcm^2)
`set_JV`	Specify a tabular data set to use as a JV characteristics per area. If the specified area does not exist it will be newly created. The command takes three arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0 3. `file-name`, Name of a file containing two columns, voltage and current density (V, $A\text{cm}^{-2}$)
`set_2DJV`	Specify a two-diode model for the JV characteristics per area. If the specified area does not exist it will be newly created. The command takes eight arguments: 1. `area-name`, Name of the area (<mesh-name>.<area-name>) 2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0 3. `J01`, Saturation current density for the first diode (ideality factor one) ($A\text{cm}^{-2}$) 4. `J02`, Saturation current density for the second diode (ideality factor two) ($A\text{cm}^{-2}$) 5. `Jph`, Photo current density ($A\text{cm}^{-2}$) 6. `Rs`, Series resistance (Ωcm^2) 7. `Rsh`, Shunt resistance (Ωcm^2) 8. `Eg`, Band gap (eV)

`set_1DJV` Specify a one-diode model for the JV characteristics per area. If the specified area does not exist it will be newly created. The command takes eight arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `J0`, Saturation current density ($A\text{cm}^{-2}$)
4. `nid`, Ideality factor
5. `Jph`, Photo current density ($A\text{cm}^{-2}$)
6. `Rs`, Series resistance (Ωcm^2)
7. `Rsh`, Shunt resistance (Ωcm^2)
8. `Eg`, Band gap (eV)

`set_R` Specify a resistance for the JV characteristics per area. If the specified area does not exist it will be newly created. The command takes three arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `R`, Resistance (Ωcm^2)

`set_T` Specify a local temperature. The command takes two arguments:

1. `area-name`, Name of the area (<mesh-name>.<area-name>)
2. `T`, Temperature (K)

`set_sel_Rel` Set an electrode resistance per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes three arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `electrode-index`, Index of the electrode. The first electrode has index 0
3. `value`, Sheet resistance value (Ω)

`set_sel_Rvp` Set the contact resistance between the positive node and an electrode per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. Together with `set_Rvn` this command allows the application of an extranal voltage. The command takes three arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `electrode-index`, Index of the electrode. The first electrode has index 0
3. `value`, Contact resistance (Ωcm^2)

`set_sel_Rvn` Set the contact resistance between the negative node and an electrode per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. Together with `set_Rvp` this command allows the application of an extranal voltage.The command takes three arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `electrode-index`, Index of the electrode. The first electrode has index 0
3. `value`, Contact resistance (Ωcm^2)

`set_sel_JV` Specify a tabular data set to use as a JV characteristics per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes three arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `file-name`, Name of a file containing two columns, voltage and current density (V, $A\text{cm}^{-2}$)

`set_sel_2DJV` Specify a two-diode model for the JV characteristics per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes eight arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `J01`, Saturation current density for the first diode (ideality factor one) ($A\text{cm}^{-2}$)
4. `J02`, Saturation current density for the second diode (ideality factor two) ($A\text{cm}^{-2}$)
5. `Jph`, Photo current density ($A\text{cm}^{-2}$)
6. `Rs`, Series resistance (Ωcm^2)
7. `Rsh`, Shunt resistance (Ωcm^2)
8. `Eg`, Band gap (eV)

`set_sel_1DJV` Specify a one-diode model for the JV characteristics per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes eight arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `J0`, Saturation current density ($A\text{cm}^{-2}$)
4. `nid`, Ideality factor
5. `Jph`, Photo current density ($A\text{cm}^{-2}$)
6. `Rs`, Series resistance (Ωcm^2)
7. `Rsh`, Shunt resistance (Ωcm^2)
8. `Eg`, Band gap (eV)

`set_sel_R` Specify a resistance for the JV characteristics per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes three arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `connection-index`, Index of the connection. The first connection, between electrode 0 and 1, has index 0
3. `R`, Resistance (Ωcm^2)

`set_sel_T` Specify a local temperature per selection, i.e. change a parameter for all elements within a selection. If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes two arguments:

1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
2. `T`, Temperature (K)

Numerical Settings

Keyword	Description
`set_SplitX`	It is sometimes useful to prevent the adaptive meshing algorithms from splitting certain nodes in x- or y-direction. This commands toggles the splitting of nodes in x-direction for a specified area (per default all nodes can be split in all directions). The command takes one argument: 1. `area-name`, Name of the area (<mesh-name>.<area-name>)
`set_SplitY`	It is sometimes useful to prevent the adaptive meshing algorithms from splitting certain nodes in x- or y-direction. This commands toggles the splitting of nodes in y-direction for a specified area (per default all nodes can be split in all directions). The command takes one argument: 1. `area-name`, Name of the area (<mesh-name>.<area-name>)
`set_sel_SplitX`	It is sometimes useful to prevent the adaptive meshing algorithms from splitting certain nodes in x- or y-direction. This commands toggles the splitting of nodes in x-direction for a selection (per default all nodes can be split in all directions). If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes one argument: 1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
`set_sel_SplitY`	It is sometimes useful to prevent the adaptive meshing algorithms from splitting certain nodes in x- or y-direction. This commands toggles the splitting of nodes in y-direction for a selection (per default all nodes can be split in all directions). If no elements are selected this command sets a parameter for all areas. If elements are selected this command may create new areas to make sure the change is confined to the selected elements. To do this it appends an area modifier name to existing area names. If the modified area name already exists the modifications are applied to this existing area. The command takes one argument: 1. `area-name`, Area name modifier (<mesh-name>.<modifier>)
`maxiter`	Set the maximum number of iterations for solving the non-linear system. The command takes one argument: 1. `maxiter`, Maximum number of iterations (default: 25)
`tol_V`	Absolute voltage tolerance for the break-off criterion. The command takes one argument: 1. `tol_V`, Absolute voltage tolerance V (default: 10^{-5}V)
`rel_tol_V`	Relative voltage tolerance for the break-off criterion. The command takes one argument: 1. `tol_V`, Relative voltage tolerance – (default: 10^{-5})
`tol_kcl`	Absolute current tolerance for the break-off criterion (KCL stands for Kirchhoff's Current Law). The command takes one argument: 1. `tol_kcl`, KCL tolerance A (default: 10^{-5}A)
`rel_tol_kcl`	Relative current tolerance for the break-off criterion (KCL stands for Kirchhoff's Current Law). The command takes one argument: 1. `tol_kcl`, Relative KCL tolerance – (default: 10^{-5})

Solving

Keyword	Description

`solve` Solve the system (do a voltage sweep). The command takes four arguments:

1. `mesh-name`, Name of the mesh
2. `V_start`, Start voltage
3. `V_end`, End voltage
4. `N_step`, Number of voltage steps

`refine_oc` Given a voltage sweep is present which includes voltages above and below V_{oc}, this command tries to refine the IV characteristics to include the open circuit point. The command takes four arguments:

1. `mesh-name`, Name of the mesh
2. `tol_i`, current tolerance
3. `tol_v`, voltage tolerance
4. `Nmax`, Maximum number of iterations

`refine_mpp` Given a voltage sweep is present of at least three points, which includes voltages above and below V_{mpp}, this command tries to refine the IV characteristics to include the maximum power point. The command takes four arguments:

1. `mesh-name`, Name of the mesh
2. `tol_i`, current tolerance
3. `tol_v`, voltage tolerance
4. `Nmax`, Maximum number of iterations

`adaptive_solve` Solve the system and adapt the mesh at one specified voltage. The command takes four arguments:

1. `mesh-name`, Name of the mesh
2. `V_a`, Applied voltage
3. `threshold`, Ralative threshold for node splitting, a parameter between 0 and 1 that controls how agressive the mesh is adapted, where lower values lead to a more agressive mesh adaption (typical values betwwen 0.3 and 0.5).
4. `N_step`, Number adaptive meshing iterations

Verbosity levels

Keyword	Description
`out_quiet`	Set verbosity to the minimum (only says something when it crashes). The command takes no arguments.
`out_normal`	Set verbosity to the normal level. The command takes no arguments.
`out_verbose`	Output additional data that may be interesting. The command takes no arguments.
`out_debug`	Output additional data that is only interesting for someone who is chasing bugs in the code. The command takes no arguments.

Misc.

Keyword	Description
`tic`	Start the timer.
`toc`	Print seconds since last `tic` command (or start of execution if no `tic` command was issued)
`define`	Define a variable. PVMOS variables only support numerical values. These variables can be used wherever a numerical value is required (see Section V B). The command takes two arguments: 1. `name`, name of the variable 2. `value`, value of the variable
`define_solpar`	Determine the solar cell parameters from the simulated IV points for a mesh and define the variables `Isc`, `Voc`, `Impp`, and `Vmpp`, with their values set accordingly. These variables can be used wherever a numerical value is required (see Section V B). The command takes one argument: 1. `mesh-name`, name of the mesh

B. Variables and expressions

PVMOS also features "expressions", which are placed between square brackets, i.e., [<expression>]. You can refer to variables, as defined with the `define` command (see the previous section), in expressions. Furthermore, PVMOS can evaluate mathematical expressions, provided PVMOS is compiled with libmatheval support. Expressions can be placed anywhere in the PVMOS input file, also as part of a word. Nesting of expressions is also allowed. Perhaps the best way to explain the use of expressions is with a few examples.
Define $N = 10$:
`define N 10`
Initialize `mesh10`:
`newmesh 0 0 1 1 [N] [N] mesh[N]`
Define some variables and calculate with them:

```
define Jph 0.03
define kT 0.0259
define J0 1e-12
define Voc [kT*log(Jph/J0+1)]
```

PVMOS with libmatheval supports the following functions:

- `exp` - exponential
- `log` - logarithmic
- `sqrt` - square root
- `sin` - sine
- `cos` - cosine
- `tan` - tangent
- `cot` - cotangent
- `sec` - secant
- `csc` - cosecant
- `asin` - inverse sine
- `acos` - inverse cosine
- `atan` - inverse tangent
- `acot` - inverse cotangent
- `asec` - inverse secant
- `acsc` - inverse cosecant
- `sinh` - hyperbolic sine
- `cosh` - cosine
- `tanh` - hyperbolic tangent
- `coth` - hyperbolic cotangent
- `sech` - hyperbolic secant
- `csch` - hyperbolic cosecant
- `asinh` - hyperbolic inverse sine
- `acosh` - hyperbolic inverse cosine
- `atanh` - hyperbolic inverse tangent

- `acoth` - hyperbolic inverse cotangent
- `asech` - hyperbolic inverse secant
- `acsch` - hyperbolic inverse cosecant
- `abs` - absolute value
- `step` - with value 1 defined for x = 0 Heaviside step function
- `nandelta` - Dirac delta function with not-a-number at $x = 0$
- `delta` - Dirac delta function with infinity at $x = 0$
- `erf` - error function

In addition libmatheval defines the following constants:

- `e` - e
- `log2e` - $\log 2(e)$
- `log10e` - $\log 10(e)$
- `ln2` - $\ln(2)$
- `ln10` - $\ln(10)$
- `pi` - π
- `pi_2` - $\pi/2$
- `pi_4` - $\pi/4$
- `1_pi` - $1/\pi$
- `2_pi` - $2/\pi$
- `2_sqrtpi` - $2/\sqrt(\pi)$
- `sqrt` - $\sqrt(2)$
- `sqrt1_2` - $\sqrt(1/2)$

Note that function and constant names cannot be redefined as a variable.

VI. EXAMPLES

In this section I discuss several examples which can be found in the example directory.

A. Monolithically series connected mini-module and a defect

In the folder `Examples/ThinFilm` you can find a PVMOS input file `thin_film8x8.mos`. This file creates a thin-film *a*-Si:H mini-module of 8x8 cm^2 with 8 cells of 1 cm wide. In addition the third cell has a defect. This example demonstrates many basic PVMOS operations such as creating new meshes, joining meshes, selecting elements, locally refine the mesh, create new area's, setting parameters for area's, and assigning elements to an area.

B. Monolithically series connected mini-module and many defects

This example consists of the files `thin_film40x40.mos` and `Shunts40x40.m`, both located in `Examples/ThinFilm`. This example depends on GNU Octave. In this example I use the file `thin_film40x40.mos` to create a 40x40 cm^2 module with 40 cells of 1 cm wide. The module that is created is defect-free. In the file `thin_film40x40.mos` no simulation is run, the script only creates a defect free mesh which is saved to a binary file. The GNU Octave script `Shunts40x40.m` creates a random distribution of defects and generates PVMOS scripts where these defects are built in. The PVMOS scripts generated by the GNU Octave script start by loading the defect-free mesh and subsequently locally refine the mesh and add shunts. This script demonstrates the simulation of larger systems (and for that reason is best used on a 64 bit operating system and a 64bit PVMOS executable and several GB of memory) and the saving and loading of meshes.

C. Metal wrap-through module

This example is located in `Examples/Pinup`. The example entails the files several files:

- `mkpvmosmesh.oct` Dynamic library for GNU Octave to write PVMOS meshes. This needs to be compiled from the PVMOS source tree ("make mkpvmosmesh") and placed in the example directory.
- `generate_mesh.m` Generates the mesh from the images listed below using GNU octave
- `pinup_frontmetal.png` Image used by `generate_mesh.m` to define the areas where there is metal at the front
- `pinup_vias.png` Image used by `generate_mesh.m` to define the areas where the front contact is connected to the contact foil through vias
- `pinup_backisolation.png` Image used by `generate_mesh.m` to define the areas where the back contact is isolating (around the vias)
- `pinup_back2contactfoil.png` Image used by `generate_mesh.m` to define the areas where the back contact is connected to the contact foil
- `pinup_foilisolation.png` Image used by `generate_mesh.m` to define the areas where the contact foil is isolating
- `pinup_contactfoil_vn.png` Image used by `generate_mesh.m` to define the areas the contact foil is connected to ground
- `pinup_contactfoil_vp.png` Image used by `generate_mesh.m` to define the areas where the contact foil is connected to the positive node.
- `pinup.mos` The PVMOS input file

This example demonstrates how a PVMOS mesh can be generated from images using GNU octave where the images are used as selection masks to select elements and change some parameters for these elements. The example simulateds a metal wrap-through solar cell with three electrose, the front electrode (emitter and front metal), the back electrode (metal contact to the base) and a contact foil at the back which contacts the front metal through vias and the back contact. After running the `generate_mesh.m` script in GNU Octave a mesh is written to `pinup.bin`. This meash is loaded in the PVMOS script `pinup.mos`. As the mesh is rather large the mesh is first simplified to reduce the number of elements in the mesh without losing resolution in he area definition. After this step the mesh is a tad coarse for simulating the potentials accurately so the mesh is refined again but this time only there where mesh refinement is needed for an accurate simulation of the potentials. After that the device IV characteristics are simulated.

D. Crystalline silicon module

The final example is perhaps the most elaborate. In `Examples/CSiSTD` we do a simulation of a crystalline solar cell with three busbars. The simulation extensively uses the definition of new meshes which are glued together to form bigger and more complicated meshes untill we have a complete cell with busbars and fingers and tabbing wires. It uses al the tricks PVMOS has to generate a complicated geometry from scratch without aid from GNU Octave. The result is a mesh that is comparatively small and therefore allows to do the simulation is a comparatively small

memory footprint (less than 0.5 GB) which makes this example even suitable for 32bit machines. There is an optional section in the input file which you can uncomment to simulate the effect of Banksy spray-painting a rat on your solar cell leaving a permanent rat-shaped shadow pattern on the cell.

---

[1] B. E. Pieters, "PVMOS: A Free and Open Source Simulation Tool for Solar Modules," *submitted to: Solar Energy Materials and Solar Cells*, 2015.

[2] T. A. Davis. (retrieved 22 May 2014) BLAS performance bug and its effect on sparse "bench" in MATLAB 7.4. [Online]. Available: `http://www.cise.ufl.edu/research/sparse/cholmod/blasbug.html`