SMARTI-

Solar Matlab Ray Tracing Implementation

*Manual:*

Inhalt

[1 Getting started 1](#_Toc510270142)

[1.1 Workflow: Performing a simulation 1](#_Toc510270143)

[2 General description of the simulation process 2](#_Toc510270144)

[3 Description of the included scripts and functions in alphabetic order 6](#_Toc510270145)

[3.1 angles2vector 6](#_Toc510270146)

[3.2 AR\_RAT.m 7](#_Toc510270147)

[3.3 detector.m 7](#_Toc510270148)

[3.4 geometry\_cube\_pyramide.m 7](#_Toc510270149)

[3.5 init\_rays.m 7](#_Toc510270150)

[3.6 input\_parameters.m 7](#_Toc510270151)

[3.7 intersect.m 7](#_Toc510270152)

[3.8 lambertsch.m 8](#_Toc510270153)

[3.9 layer.m 8](#_Toc510270154)

[3.9.1 Skipping side intersections => “Booster” 8](#_Toc510270155)

[3.9.2 Implementation of finger and busbar reflection and absorption 9](#_Toc510270156)

[3.10 load\_data.m 10](#_Toc510270157)

[3.11 SMARTI\_MAIN.m 10](#_Toc510270158)

[3.11.1 Calculating solution path for generation profiles 10](#_Toc510270159)

[3.12 snell.m 11](#_Toc510270160)

[4 Analyzing tools 11](#_Toc510270161)

[4.1 Overview\_processes\_oneAOI.m 11](#_Toc510270162)

[4.2 From\_G\_to\_j.m 11](#_Toc510270163)

[5 Literaturverzeichnis 12](#_Toc510270164)

# Getting started

## Workflow: Performing a simulation

1. Modify parameters in file **input\_parameters.m**
   1. Modify parameters as desired. The following parameters can include multiple values to perform batch simulations:  
      **- alpha\_batch  
      - theta\_batch  
      - d\_AR** (only one of the cells can contain an array with multiple inputs) **- p\_bot\_batch** (only one of the cells can contain an array with multiple inputs) **- d\_layer\_batch** (only one of the cells can contain an array with multiple inputs)
   2. The number of layers is set to five as standard. Adding or reducing the number of layers can be done by adding or subtracting a value to the array of the following parameters:  
      **- layer\_mat  
      - AR\_mat  
      - d\_AR  
      - S\_geom  
      - p\_bot  
      - d\_layer  
      - lambert  
      - grid\_pos  
      - detect\_bot  
      - detect\_top**
   3. If one of the desired materials is not in the list of materials that are loaded regularly in the **Mat** variable the excel sheet **All\_data\_PV-Lighthouse.xlsx** in the data folder can be opened to check if the desired material can be found there. To include the material either an unused material can be replaced or additional Mat inputs can be created. The specification for the new material needs to follow the form: **{'GeneralName', 'SpecificName', ‘[Source]’}**
2. Run the **SMARTI.m** function and choose the location were the results shell be stored (e.g. the results folder in the SMARTI program path).
3. Use the corresponding functions in the **Analyzing-tools** folder to plot the simulated data
   1. Plot the distribution of the reflection, absorption and transmission processes
   2. Generate the EQE and the IQE curves using the function **from\_G\_to\_j.m** (the corresponding PC1D software needs to be downloaded and the path added to the function. For more details read section 3.13.1.)

# General description of the simulation process

Ray tracing algorithms are of particular interest for engineers and scientists working on realistic image processing. Rendering static images or moving scenes is mainly done by ray tracing because no other computational technic leads to a more realistic appearance (Glassner 2007). Even though for image processing some additional features are applied (rasterization, shading, anti-aliasing …) and the starting as well as the detection objects differ (in computer graphics, rays generally leave from every pixel leading to the light sources, which is referred to as backward ray tracing) (Sherrod 2008), the basic principles that are used in computer graphics also apply to photovoltaic ray tracing.

In the Monte Carlo method, a numerical approach is defined that uses a random number generator for the determination of a physical process, which itself is determined by random processes or is of such high complexity that analytical procedures are not feasible (Holst 2015). The method of Monte Carlo modeling is widely used in many research fields and is, e.g., an essential component of experimental particle physics at CERN (European Organization for Nuclear Research) (Maria Grazia Pia und Georg Weidenspointner 2012).

In the case of ray tracing, the application of the Monte Carlo method is also referred to as Monte-Carlo particle tracing (Holst 2015), where many rays that represent photons are individually traced until they interact with a surface. Here, depending on the number that is calculated by the random number generator, a decision is made as to what is happening to the ray. In solar cell ray tracing tools, the decision will generally be if it is absorbed, reflected, or transmitted. Additional decisions such as if the reflection will be specular or diffuse can be taken using the same random number or by the creation of another one.

It would be computationally very costly to create a model involving all the pyramidal structures of a solar cell and subsequently perform a ray tracing study on the model, which is why a so-called unit cell approach is mainly used for the simulation. The unit cell is a cuboid (or any other volume element that can be laterally ordered without a gap) that incorporates all the structures that repeat periodically. If a ray hits the side of the unit cell, it will receive a new starting point which is situated to the opposite of where the side was intersected. The direction of the ray is conserved. Like this, by using symmetry, an infinite array of pyramids can be modeled with little effort. In Fig. 3.6 an example is plotted for a SMARTI simulation of a solar module with textured front side encapsulant and a textured solar cell (layer thicknesses are not to scale).

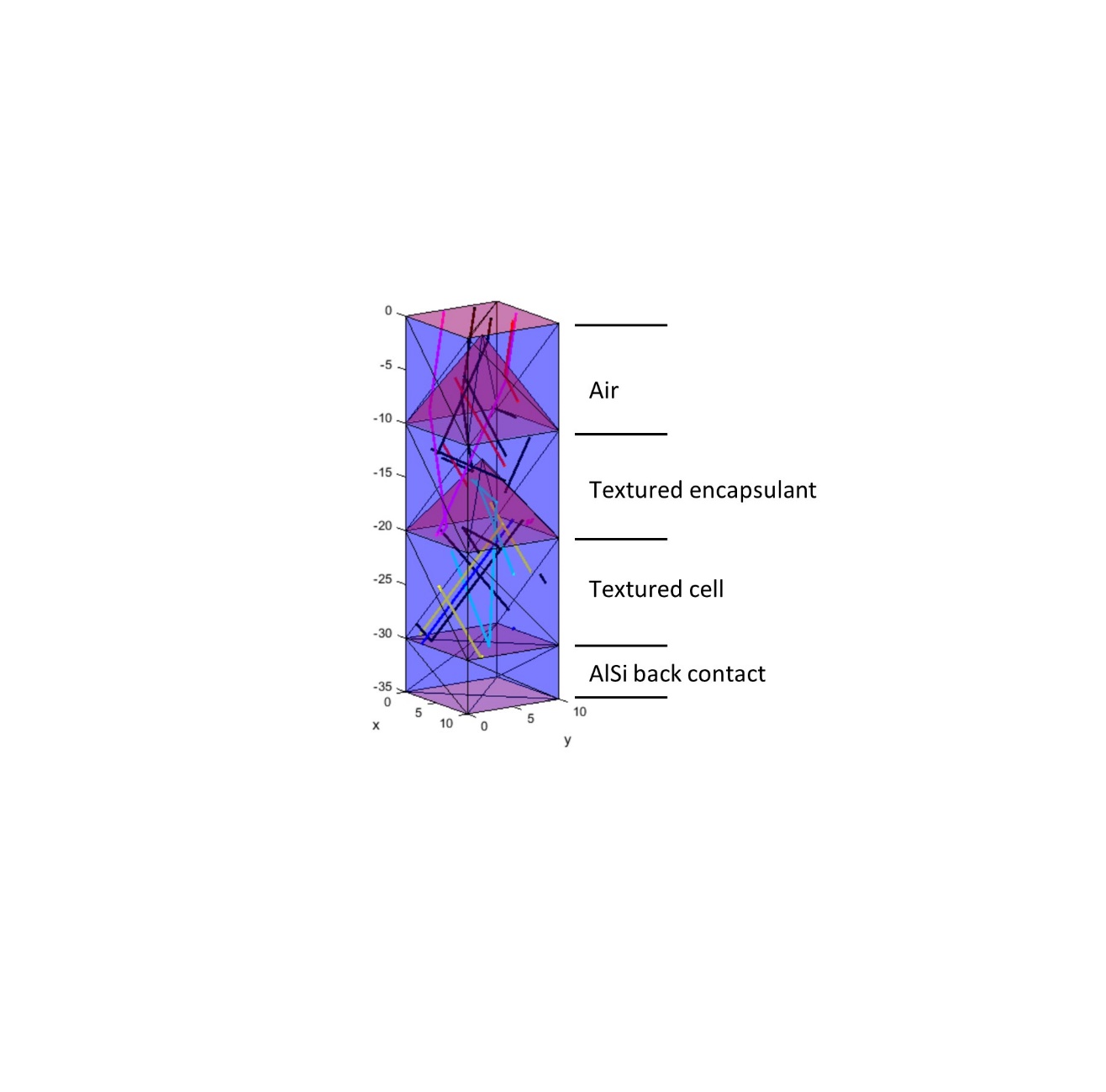


Fig. 3.6: Unit cell of a solar module simulated in SMARTI consisting of 4 layers (air, encapsulant, Si, AlSi). A pyramidal texture is added to the top of the second and the third layer. The heights of the individual layers are not to scale.

The central element of the software is a function representing any layer that can be implemented in a solar module. The function input consists of a number of input parameters, which define the properties of the layer. These parameters are

* + The position of the layer in the layer stack
  + The material and thickness of the AR coating if one is applied
  + The geometry of the layer including its size and the texture on its bottom
  + The material the layer is consisting of
  + The grid consisting of contact fingers and busbars if one is applied
  + The number of maximum bounces the rays may perform inside the layer
  + Plotting parameters for a graphical representation of the rays inside the stack of layers

Apart from this of course, the numbers of rays that enter the layer from the top and from the bottom with their starting point and their orientation are a central input element for the function. The outputs of the function are then a list of rays leaving the slab from the top and from the bottom including their position and their orientation and a list of rays that were absorbed inside the slab with the position of absorption. Additional output parameters are if any ray was lost as no intersection point could be found or if the calculations were interrupted because of the maximum number of bounces. A schematic representation of the function is given in Fig. 3.7.

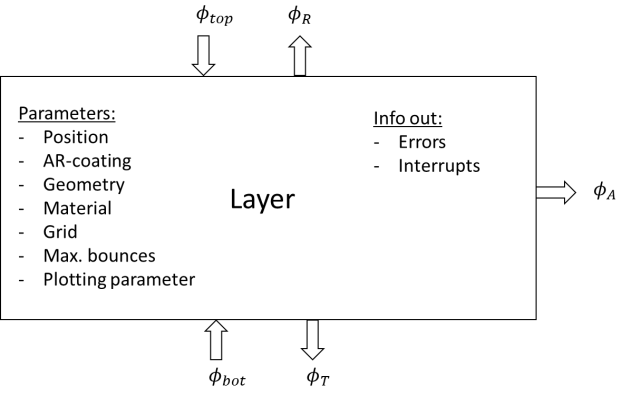


Fig. 3.7: Schematic representation of the Layer function in the SMARTI ray tracer

Having clearly defined the layer-function, the desired number of layers can be stacked on top of each other. Then, using a while loop, one layer after the other starting from the top is calculated using the output rays of the adjacent layers as input rays for the calculation. When the calculation of the last layer in the stack is performed, the while loop restarts with the first layer until all the rays are either transmitted through the stack, absorbed inside the layers, or reflected back out of the top layer. This process is depicted in Fig. 3.8.

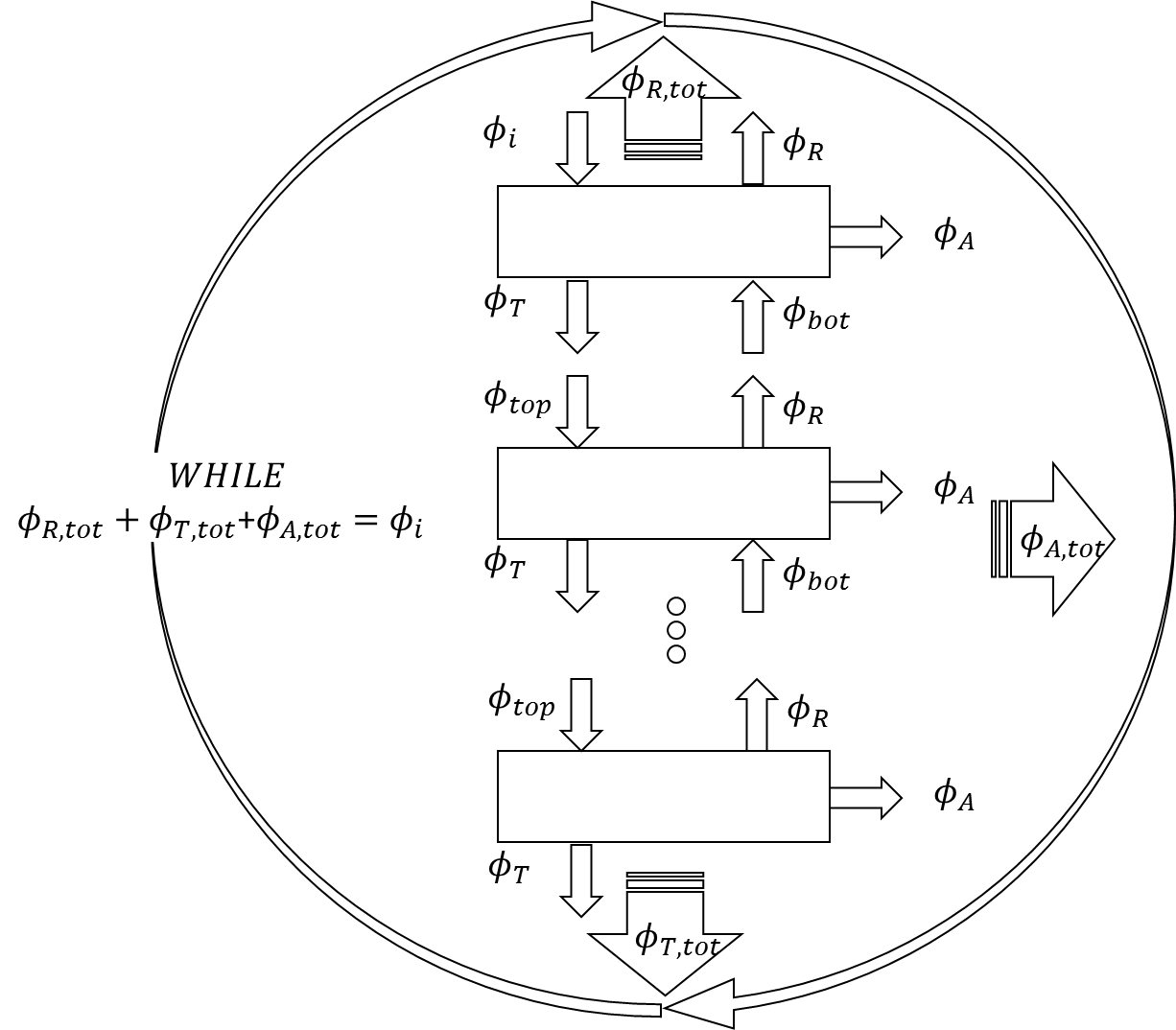


Fig. 3.8: Schematic representation of the ray tracing algorithm that is implemented in the SMARTI ray tracer

Inside the layer function, a mathematical algorithm is implemented that calculates the intersection points of the rays with the infinite planes that are formed by the geometry. After the determination of the intersection point, it is tested if the point lies within one of the triangles defining the geometrical structures.

To execute these two steps, it is necessary to give a mathematical definition for the rays and for the triangles that are involved in the calculation. There are multiple options to proceed concerning these definitions. The one chosen here gives a parametric definition for any point either on the ray or on the surface.

The parametric definition defines any point on a line by the vector that starts in the origin of the coordinate system. It is given by

|  |  |  |
| --- | --- | --- |
|  |  | (138) |

where is a unit vector indicating the direction in which the ray is travelling and is the distance between the starting point .

Each of the triangles, which together form the geometry of the layer, can be described by three 3 x 1 vectors. The first one is the position vector indicating one corner of the triangle. The two other vectors and define two sides of the triangle and thus implicitly also the third side. It is important to note that the top of the surface is also defined by the order of these two vectors. The cross product of the two vectors results in a vector that indicates the direction in which the surface points and thus can be hit by a ray. Now any point on the whole plane that is defined by the triangle can be written in parametric form as

|  |  |  |
| --- | --- | --- |
|  |  | (139) |

At this point it is important to realize that this forms a coordinate system, which has its origin at and the direction of its two axis are formed by the vectors and . The variables and are the indicators of how many steps of the units with length and to be followed to come to point . To realize this is crucial to understand the steps that are needed for the decision if the intersection is inside or outside of the triangle. Coordinate systems that are defined in such a way are called *barycentric coordinate systems* (Shirley et al. 2010).

The definition of the vectors can be seen in Fig. 3.9.

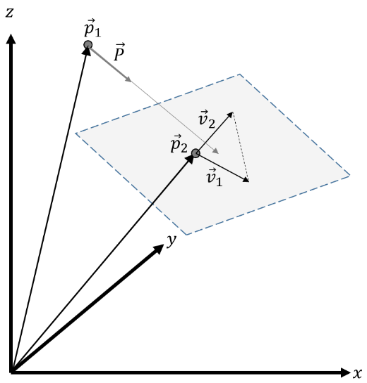


Fig. 3.9: Definition of the position and the direction vectors describing the ray and the geometry of the object

To find the intersection of the ray with the plane, the two parametric equations are set equal so that the following is received.

|  |  |  |
| --- | --- | --- |
|  |  | (140) |

Rearranging this term gives

|  |  |  |
| --- | --- | --- |
|  |  | (141) |

In coordinate representation, this is written as:

|  |  |  |
| --- | --- | --- |
|  |  | (142) |

Solving this simple system of linear equations is a very crucial step for every ray tracing software, as it consumes a very high share of the total simulation time. Even though many software tools such as MATLAB are capable of solving such a system in one step, it is often reasonable to manually program this step as the time consumption can be reduced.

A significant reduction of run-time could be achieved by the implementation of Cramer’s rule (Poole 2015). This standard method for solving systems of linear equations can be tuned to increase the solving speed as the results of individual mathematical operations can be reused in different parts of the calculation procedure. The detailed description of the method can be found in (Shirley et al. 2010).

When the variables , and are found, the next step is to decide whether the intersection point of the ray with the surface is inside or outside of the triangle. As described above, the parameters and can be seen as linear factors describing any point in a coordinate system, which is built up by the basis vectors and . The triangle of interest is the one formed by and . Thus, the first thing that has to be determined is whether and are greater than 0. Another condition that needs to be fulfilled to be sure that the intersection point is inside the triangle is that the sum of and needs to be smaller than unity (Shirley et al. 2010).

It could also be that the intersection point lies behind the starting point of the ray, which would result in a negative value for . This would also be a reason to reject the intersection with the triangle. The final test for the correct determination of the intersection algorithm is that exactly one triangle may be hit during each run If more or fewer triangles are hit, an error is displayed.

When the right intersection point is found, a random number between zero and one is generated and the reflectivity , transmissivity , and absorptivity for the given angle of incidence and the given material parameter are calculated. If , then the ray will be reflected and if , the ray will be absorbed. In all other cases, the ray will be transmitted.

The ray tracing software includes many additional features, such as grid implementation, Lambertian scattering, a speed boost function, and isotropic irradiation that are described in detail in the following sections.

# Description of the included scripts and functions in alphabetic order

In the following the scripts and functions that the SMARTI software consists of are explained in detail.

## angles2vector

**Type:** function

**Inputs:** (theta, alpha)  
Where: theta= angle of incidence; alpha= azimuth angle (both in °)

**Outputs:** Corresponding vector to the both angular inputs

**Explanation:** This function creates a unit vector to the corresponding inputs angles. The direction of the vector will always be in the negative z-direction. The 0°-position of the azimuth angle is the x-axis.

## AR\_RAT.m

**Type:** function

**Inputs:** (complex\_n1, complex\_nAR, complex\_n2, wavelength, d\_AR, theta);   
Where: complex\_n1=Complex refractive index of incoming medium; complex\_nAR=complex refractive index of coating; complex\_n2=complex refractive index of outgoing medium; wavelength=wavelength of the incoming light; d\_AR=thickness of coating; theta=angle of incidence

**Outputs:** [R,T,A]   
Where: R=Reflection, T=transmission, A=absorption

**Explanation:** The algorithm is taken from (Macleod 2001) using the transfer matrix method. In principle it can be used to include multiple thin layers but here only one layer is foreseen.

## detector.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

## geometry\_cube\_pyramide.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

## init\_rays.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

## input\_parameters.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

## intersect.m

**Type:** bla

**Inputs:** (p,P,S,rst\_struct\_init)  
Where: p: Vector indicating start position of ray; P: Vector indicating direction of ray; S: Struct including fields for triangles (corner point: p, triangle side 1: v1, triangle side 2: v2, normal vectors N, positions: pos)

**Outputs:** [rst\_struct hit]

**Explanation:** bla

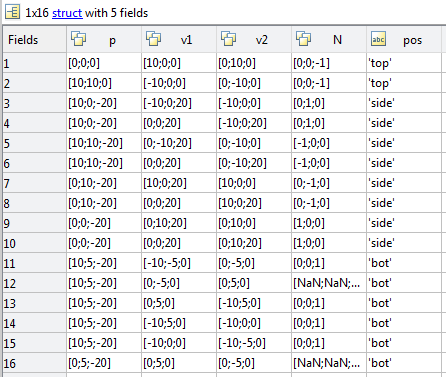


Figure 1: Example for a the S-struct containing the triangles of the layer geometries

## lambertsch.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

## layer.m

**Type:** bla

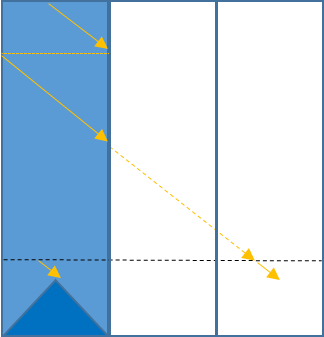
**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

### Skipping side intersections => “Booster”

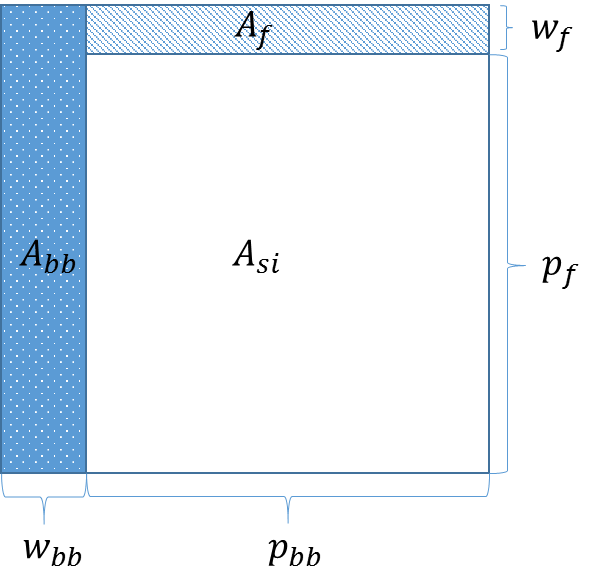
* It can be switched on or off in layer.mat => Booster=1 or 0;
* The algorithm starts when there is a second side hit (side\_hit==2)
  + side\_hit is reset at every hit of the top or bottom
* A layer is defined, that is located 0.1 µm above the top of the pyramid
  + For the inverse case (ray pointing upwards) the layer is defined to be 0.1 µm beneath the top pyramide



* The rst values for the ray/layer intersection are then calculated
* It is then checked if the ray is absorbed and if so the z position of the absorption is stored
* If the ray is not absorbed, the hit point is projected back into the unit cell

### Implementation of finger and busbar reflection and absorption

* Calculation of the finger and bus bar share of the total module area



Share of busbars:

Share of fingers:

Storing grid position, grid share and grid material:

grid.pos={0 0 1 0}; %On top of the layer

grid.share={Mbb Mf};

grid.mat={Ag Al};

In the layer function at every top and bottom intersection the same algorithm is applied

Two random numbers are created.

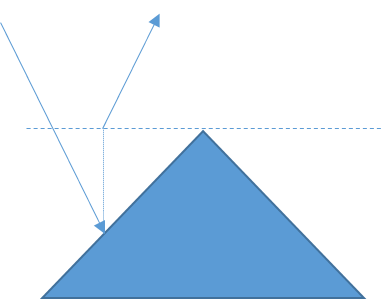
The first is used to decide if the ray hits the finger, the busbar or the cell.

The second is then used in the case of a finger or the busbar being hitten, to decide if the ray is reflected or absorbed in the material.

The cases of hitting the fingers or busbars from above or from beneath is differentiated.

Case ‘top’: The reflection angle is calculated by ‘angle of reflection equals angle of incidence’

Case ‘bot’: The reflected ray is projected on the horizontal plane at the peak of the texture. There the angle of reflection is calculated as if the ray were hitting a horizontal plane.



It is checked if the ray after hitting the texture is reflected again onto the same texture. If this is the case, the grid algorithm is inactivated.

## load\_data.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

## SMARTI\_MAIN.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

### Calculating solution path for generation profiles

* The solution path is different from the dimension z for 1/6 the pyramide hight
* Two different regions for path calculation are distinguished

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

## snell.m

**Type:** bla

**Inputs:** bla

**Outputs:** bla

**Explanation:** bla

# Analyzing tools

The tools in the folder ‘Analyzing tools’ are explained in the following.

## Overview\_processes\_oneAOI.m

This script can be used to plot all the optical absorption, transmission and reflection processes at one angle of incidence.

## From\_G\_to\_j.m

To be able to use this script the following steps need to be performed:

1. The command line version of PC1D (‘PC1Dmod and cmd-PC1D v6.2.2.zip’) needs to be downloaded from   
   <https://www2.pvlighthouse.com.au/resources/PC1D/PC1Dmod6/PC1Dmod6.aspx>
2. The folder needs to be unzipped and stored to any desired location
3. Copy the file ‘PVcell\_simple.prm’ and rename the copy to ‘PVcell\_GenBatch.prm’
4. Copy the path of the folder where the .prm document is located and replace the path in the function ‘From\_G\_to\_j.m’ where indicated (directly after the first for-loop is initialized).
5. Now the electrical properties of the ‘PVcell\_GenBatch.prm’ can be modified with the PC1D5.exe

Literaturverzeichnis

Glassner, Andrew S. (2007): An introduction to ray tracing. digital print. San Francisco, Calif.: Kaufmann.

Holst, Hendrik (2015): Development and application of a modular ray tracing framework to multi-scale simulations in photovoltaics. Gottfried Wilhelm Leibnitz Universität Hannover, Hannover.

Macleod, H. A. (2001): Thin-film optical filters. 3rd ed. Bristol, Philadelphia: Institute of Physics Pub (Series in optics and optoelectronics).

Maria Grazia Pia; Georg Weidenspointner (Hg.) (2012): Monte Carlo Simulation for Particle Detectors. CERN Council Open Symposium on European Strategy for Particle Physics. Krakow.

Poole, David (2015): Linear algebra. A modern introduction. 4th ed. Stamford, CT: Cengage Learning.

Sherrod, Allen (2008): Game graphics programming. Boston, MA: Course Technology/Charles River Media/Cengage Learning (Course Technology PTR game development series).

Shirley, Peter; Marschner, Stephen Robert; Ashikhmin, Michael (2010): Fundamentals of computer graphics. 3. ed., [Nachdr.]. Natick, Mass.: A K Peters.