Plasmonic Light Trapping Transforming Thin-Film Photovoltaics

Keyur Gandhi and Professor S. Ravi P. Silva, Advanced Technology Institute, Department of Electronic Engineering, University of Surrey, Guildford GU2 7XH, United Kingdom

The photovoltaics (PV) market represents the fastest growing renewable energy technology, with a 35% annual growth rate, which is projected to be worth $150B by 2018. The global cumulative installed PV capacity has exceeded 140 GW in 2013 and is generating 160 TWh/year of electricity, representing 0.85% of worldwide electricity demand. At the present growth rate, it is projected that over 20% of all electricity could be supplied by PV by 2050.

PV has evolved through three generations of device technology, where each subsequent generation has focused on achieving a lower power generation cost per watt through the use of thinner semiconducting materials on low cost substrates. The 1st generation (1G) of mono- and multicrystalline silicon technology was followed by 2nd generation (2G) thin-film PV cells, such as amorphous and polycrystalline silicon, a technology sector growing steadily due to its lower cost and material usage. The majority of the cost associated with fabricating PV modules is due to the semiconducting material, with another significant fraction associated with mounting onto structures. Reducing the volume of the semiconducting layer in PV devices will lower the module cost for 2G PVs, but they are less efficient in converting incident light to electricity. The 3rd generation (3G) PV technology is targeting further reductions in manufacturing cost while maintaining the power conversion efficiency of the modules through the integration of nanotechnology.


Plasmonic nanomaterials are metal nanoparticles, typically of gold or silver, which scatter light in the semiconducting layer and can significantly increase light absorption in the PV cell. The new design approach allows PV manufacturers to simulate these plasmonic-enhanced thin-film PV cells prior to fabrication, and allows them to examine the optimum design architecture. Finite Difference Time Domain (FDTD) method in the Atlas Luminous framework was used to examine the plasmonic effects on the performance of PV devices. The plasmonic nanomaterial, due to their optical properties, also acts as a broadband anti-reflection coating to increase light coupling to PV cells.

Continued on page 2...
Figure 1 shows a graphical representation of the plasmonic effect of the metal nanoparticles in the PV device under illumination. Such nanoparticles can be easily integrated in the PV device architecture during the fabrication process.\cite{3} Light interaction with the plasmonic structures induces plasmons, i.e. collective surface oscillations of conduction electrons in the metal nanoparticles, which traps light by either exciting localised surface plasmon resonance (LSPR) (Figure 1a) or light scattering (Figure 1b).

Coupled opto-electronic simulations were performed on thin-film amorphous silicon PV devices with gold metal nanoparticles of 30 nm radii used to demonstrate the plasmonic effects in different positions in the device structure (Figure 2). More detailed information on the simulation and parameters used can be found in the published article.\cite{4} Incident light (AM1.5G solar spectrum) was used to perform opto-electronic simulations and to calculate current density-voltage (J-V) curves,

![Figure 1](image1.png)

Figure 1. (a) Light concentration by the excitation of localised surface plasmons resonance around metal nanoparticles embedded in the semiconductor of PV device. (b) Light scattering by the metal nanoparticles at the surface of the PV device. The position of the P/N junction is indicated by the solid black line in the active layer of the PV device.

![Figure 2](image2.png)

Figure 2. (a-c) PV cell device structures with a 200 nm thick a-Si:H active layer: (a) reference and plasmonic device structures incorporating 30 nm radii gold nanoparticles embedded between the (b) ITO/a-Si:H and (c) ITO/SiO2 layers respectively. (d-f) Optical Intensity profile calculated from FDTD analysis under AM1.5G illumination. (g-i) Photogeneration rate calculated from the optical intensity profile plotted in the a-Si:H layer. (j-l) Electric field intensity calculated from the optical intensity and the built-in field in the a-Si:H layer. (m-o) Recombination rate profile in the a-Si:H layer. The scale bar in each case = 100 nm.
photogeneration and recombination rates, as well as providing spectral response data such as surface reflection, absorption and transmission. The syntax used for the definition of the incoming beam is defined as:

```
Beam num=1 x.o=0.0 y.o=-25.0 angle=90.0 AM1.5 wavel.start=0.295 wavel.end=0.805
wavel.num=51 FDTD fd.auto td.srate=5 prop.leng=150.0 s.top big.index te cos
phase=0.0 td.errmax=0.01 dt=5.0*E-18
td.end periodic verbose
```

Whereas, the syntax used to define the phase matched layers is described as:

```
pml top degree=1 width=25.0 r90=0.01
pml bottom degree=1 width=25.0 r90=0.01
```

The material parameters of the amorphous silicon are defined as follows:

```
material material=silicon mun=1 mup=0.05
nc300=2.0*e21 nv300=1.0*e22 eg300=1.75
affinity=4.0 taun0=1.0*E-6 taup0=1.0*E-6
```

The defects states in the bandgap of amorphous silicon are defined as follows:

```
defects nta=1.e21 ntd=1.e21 wta=0.033
wt0=0.049 nga=1.5*e15 ngd=1.5*e15
ega=0.62 egd=0.78 wga=0.15 wgd=0.15
sigtae=1.e-17 sigta0=1.e-15 sigtde=1.e-15
sigtdh=1.e-17 siggae=2.e-16
siggae=2.e-15 siggde=2.e-15 siggdh=2.e-16
tfile=Defects.log continuous
```

Steady state responses such as optical intensity [Figure 2(d-f)], photogeneration rate [Figure 2(g-i)], electric field [Figure 2(j-l)] and recombination rate [Figure 2(m-o)] were obtained as a result of the simulation. These steady state responses assist in the optimisation of the plasmonic effects, both electrical [Figure 1(a)] and optical [Figure 1(b)], within the thin-film PV cells. The initial results suggest at least a 5% increase in the performance of the PV device can be achieved upon the incorporation of plasmonic structures. Such plasmon nanomaterial technology can also be applicable to other thin-film technologies such as poly-crystalline silicon, cadmium telluride (CdTe) and copper indium gallium selenide (CIGS), as well as organic photovoltaics, with similar gains in efficiency expected. The code produced in the collaboration between the University and Silvaco is expected to become a valuable asset to the future design of nanoparticle enhanced PV technology.

**Reference:**

How can I develop my own etch model in Victory Process?

**Background:** Victory Process is packaged with a wide range of etch models. Victory Process also supports an “Open Modelling Interface” (OMI). The OMI enables users to develop their own models inside Victory Process. Both etch and deposition models can be defined by the user in the OMI.

**Example:** In this example we are going to demonstrate how to define your own etch model.

Shown in Figure 1 is the starting structure.

The demonstration structure is a simple piece of silicon with uniform geometry in the y direction and simple stepped surface.

The etch model is a CMP style model which will control the etch rate as a function of distance from the surface. As such, the surface high point (left hand side of the structure) will etch at a higher rate than the lower point of the surface (right hand side of the structure).

The C interpreter file / code to create the etch model is shown below

```c
/**********************************************************************
* cmp_physical
* Comment: rate dependent on point's
* distance to the
* top surface point
***********************************************************************/
int cmp_physical(double max_z,
                 double x, double y,
                 double z,
                 double r0,
                 double ratio0,
                 double* rate)
{
    if ( z >= max_z )
    {
        *rate = r0
    }
    else
    {
        *rate = r0 -( max_z - z) * ratio0;
    }
    return 0;
}
```

The first section of the file defines the inputs and output of the file.

Here the inputs are:
- the z-coordinate of the highest point of the surface \( (max_z) \)
- the \( x, y, z \) location of the surface point
- the maximum etch rate, \( r0 \). This is set in the deck (defined in the material characteristics)
- the decrease of the etch rate with the distance from the highest surface point, \( ratio0 \), again set in the deck (defined in the material characteristics)
- the value returned to the simulator is \( *rate \). This is the etch rate at point \( (X/Y/Z) \)

Victory Process will call this C-function for every surface point (once every time step) of the simulated structure.

The last part of the file manipulates these inputs and returns the output. In this example we assume that the etch rate decreases linearly with the distance from the highest point of the surface: \( *rate = r0 -( max_z - z) * ratio0 \)
The deck used in this example is shown below:

```
GO victoryprocess

INIT material="silicon" from="0, 0" \
to="2, 0.2" depth=1.5 \
gasheight=0.5 resolution=0.08 \
flow.dim=3d meshDepth=2

SPECIFYMASKPOLY maskID=2 \
  istansparent=false \
p="1, -5" p="1, 5" p="5, 5" p="5, -5"

ETCH material=silicon thickness=0.5 \
  maskID=2 max

REACTION name=my_name \
  c_function=cmp_physical \
  depend1=highestSurfaceCoordinate \
  depend2=position \
  depend3=rate \
  depend4=rate

TOPOGRAPHYMODEL name="my_etch_model" \
  reactionmodel=my_name \
  fluxmodel="constant"

ETCHDEPOPROPERTIES \
  chemicalname="etcherrate" \
  material="silicon" rate=1 ratio=0.3

ETCH model="my_etch_model" \
  chemicalname="etcherrate" time=0.30

SAVE name="after_cmp"
EXPORT structure="after_cmp_ex.str"

QUIT
```

The first part of the deck is quite conventional, setting up the starting material. The REACTION statement first gives a name to the reaction model that is to be created and then instructs Victory Process to use the C Interpreter function shown earlier. Finally this statement lists the dependencies (DEPEND<n>) which are the set of parameters sent to the C-function.

The topography model is then flagged, named and set to call the reaction model as well as defining the flux model.

Finally the etch can be performed, the etch is initiated by the ETCH statement. This calls in the etch machine that has been defined in the previous statements.

Shown in Figure 2 is the product of this executed deck. Shown in Figure 3 is a 2D overlay of slices taken through the structures shown in Figure 1 and 2. The etch model used here is dependent on the depth from the surface. As distance increases into the substrate, the etch rate decreases. As such the left hand side of the structure (the highest point) has etched significantly faster than the lower area on the right hand side.

The etch feature is not limited to a single material or a pseudo2D block (uniform geometry in the y direction). For example, shown in Figure 4 is a structure with an elliptical trench in the silicon into which concentric rings of nitride and polysilicon have been added. Victory Process can call in external mask files (GDS2), however in this instance the masks used to create these shapes were specified inside the Victory Process deck.

The silicon, nitride and polysilicon are then subjected to the distance etch rate model introduced previously. The
different materials are assigned different rate characteristics so the polysilicon is the least etched material, followed by the nitride and finally the material to be etched the fastest is the silicon. As the distance rate etch model is used on all three materials, the higher points are etched fastest and the lower points more slowly.

**Conclusion**

In this example we have demonstrated one of the advanced features of Victory Process, namely the Open Modelling Interface and its application for the development of user defined etch models via a CMP style model.
**Hints, Tips and Solutions**

**How can I Crop and Slice in the Victory Process 3D Exports?**

This document is a short guide on the new and existing support for export cropping/slicing in Victory Process cell mode. The cropping operation is useful to extract a segment of the cell mode structure for further simulation. For example, a smaller subsection of a structure can be extracted, or a polygon mask crop can be used to extract a non-axis aligned segment. This allows an exported structure to be non-cuboid.

The slicing operation is similar to cropping, yet both the structure inside the sliced section, and outside is retained. After the slice is complete, the regions now outside the sliced region are relabeled with unique region numbers to those inside. This operation is useful to partition an exported structure into multiple regions, allowing easy visualization of the inside of the structure in Tonployt3d. It is also useful to set different properties for different regions, while maintaining the same materials, within a device simulation.

**Introduction**

Sections 1 and 2 illustrate the cropping support using boxes and masks, respectively. The existing box crop is demonstrated and some new examples of cropping (both boxes and masks) applied to the atlas and victory conformal cell mode exports are given. Section 1 also includes an example of z-axis coordinates for the box cropping. Mirroring is shown for some examples to clarify the cropping/mirroring execution order.

Sections 3 and 4 cover the box/mask slicing support. The slicing operation is identical to cropping, with the exception that both the outside and inside parts of the structure are retained; the outside part is given unique region numbers. Mirroring is supported for all slicing operations.

Figure 1 shows our input structure. The structure has a bounding box of “5.0, 5.0, -6.0” to “8.0, 8.0, 1.0”, and contains four materials in five regions (a substrate electrode is specified on the xy plane at z = 1.0).

**1. Crop Box**

**Supported Exports:** All cell mode exports

In this operation, a box (that falls inside the input structures’ bounding box) can be used to crop part of the structure. Figure 2 shows an example of the result from a crop box for the xy plane from “5.25, 5.25” to “7.25, 7.25”. The part of the structure outside the crop box is removed.

Syntax:

```
export victory(delaunay) \
  structure="delaunay_crop_box_2d.str" \
  crop.from = "5.25, 5.25" \
  crop.to = "7.25, 7.25"
```

A box cropped structure can also be mirrored. Figure 3 shows an example of mirroring the crop box “5.25, 5.25” to “7.25, 7.25” in the atlas(conformal) export. Note how the positive x-plane for the mirroring operation is now at x = 7.25. The regions outside are removed before the mirroring operation.

![Figure 1. Original input sicex10_2_0 round structure. left: materials, right: regions.](image-url)
Figure 2. Victory(delaunay) crop by box.

Syntax:

```
export atlas(conformal) \\
  structure="conformal_crop_box_2d_mirror_x.str" \\
  crop.from = "5.25, 5.25" \\
  crop.to = "7.25, 7.25" \\
  mirror= "+x"
```

Figure 3. Atlas(conformal) crop by box and mirrored in +x.

Syntax:

```
export victory(conformal) \\
  structure="conformal_crop_box_2d.str" \\
  crop.from = "5.25, 5.25" \\
  crop.to = "7.25, 7.25"
```

Figure 4 shows an example of the crop box operation for the victory(conformal) export including a z-value, where the regions inside the box from “5.6, 6.0, -5.5” to “7.4, 7.7, -2.0” are retained.

Syntax:

```
export victory(conformal) \\
  structure="conformal_crop_box_3d.str" \\
  crop.from = "5.6, 6.0, -5.5" \\
  crop.to = "7.4, 7.7, -2.0"
```
2. Crop Mask

**Supported Exports**: victory(delaunay), victory(conformal), atlas(reduced), atlas(conformal)

Similarly to the box crop, a convex mask can be specified as the cropping boundaries. This mask must be within the bounding box of the input structure and is specified for any xy-plane. In general, mirroring is not supported, unless the mask edges are all axis aligned. Figure 5 shows an example of the mask crop for the atlas(conformal) export. The regions inside the mask specified by “m1” are kept. The regions outside are removed. The syntax crop. maskID = ... can also be used.

**Syntax**:

```plaintext
Specify maskpoly maskname="m1" 
  P1="6.81, 5.55" \ 
  P2="5.69, 5.91" \ 
  P3="5.69, 7.09" \ 
  P4="6.81, 7.45" \ 
  P5="7.5, 6.5"

eexport atlas(conformal) \
  structure="conformal_crop_mask.str" \
  crop.mask="m1"
```

Figure 4. Victory(conformal) crop by box.

Figure 5. Atlas(conformal) crop by mask.
Figure 6 demonstrates a further example of mask cropping. In this case the victory(conformal) export is cropped. Note how the round SiO2 slope is stepped in the atlas conformal export (Figure 5), but not victory(conformal).

Syntax:

\[
\text{Specifymaskpoly maskname="m1" } \\
P1="6.81, 5.55" \ \\
P2="5.69, 5.91" \ \\
P3="5.69, 7.09" \ \\
P4="6.81, 7.45" \ \\
P5="7.5, 6.5"
\]

\[
\text{export victory(conformal) } \\
\text{structure="conformal_crop_mask.str" } \\
\text{crop.mask="m1"}
\]

3. Slice Box

Supported Exports: All cell mode exports

In this operation, similarly to the crop box, the syntax slice.from and slice.to can be used to specify a sub-section of the structure. In comparison to cropping, slicing will retain the parts of the structure outside the given box in addition to the parts inside. The regions outside the slice box will be given different region numbers to the regions inside. Figure 7 shows the regions inside the slicing box from “5.25, 5.25” to “7.25, 7.25”. In comparison to Figure 2 the outside regions are also kept but relabeled with unique region numbers. Note how the mirroring is performed for the x-plane at x = 8 instead of x = 7.25 as in Figure 2 (since the entire structure is retained by the slicing operation).
Figure 8. Victory(delaunay) slice by box.

Syntax:

```plaintext
export atlas(conformal) \
    structure="conformal_slice_box_mirror_x.str" \
    slice.from = "5.25, 5.25" \
    slice.to = "7.25, 7.25" \
    mirror="+x"
```

A further example of the box slicing operation applied to the victory(delaunay) export is shown in Figure 8. The outside regions are kept and given unique region numbers. The outside regions are shown meshed, revealing the bottom of the sliced region \( z = -2.0 \).

Figure 9. Victory(conformal) slice by mask.

Syntax:

```plaintext
export victory(delaunay) \
    structure="delaunay_slice_box.str" \
    slice.from = "5.25, 5.25, -6.0" \
    slice.to = "7.25, 7.25, -2.0"
```

4. Slice Mask

**Supported Exports:** All cell mode exports.

The slice mask operation can be used to specify a convex mask on an xy-plane to ‘slice’ the input structure. Similarly to the crop mask operation, the input structure is split into two pieces, yet the outside piece is now kept and relabeled with unique region numbers.
Figure 9 shows an example of the hexagon mask slicing the input structure from Figure 1 for the victory(conformal) export. The outside regions have been shown meshed to illustrate the region boundaries at the slicing mask planes.

Syntax:

```
Specifymaskpoly maskname="m1" \\
P1="6.81, 5.55" \\
P2="5.69, 5.91" \\
P3="5.69, 7.09" \\
P4="6.81, 7.45" \\
P5="7.5, 6.5"
```

```
export victory(conformal) \\
structure="conformal_slice_mask.str" \\
slice.mask="m1"
```

Figure 10 shows an example of the hexagon mask applied to the atlas(conformal) export resulting in separate regions inside the sliced convex hull. Note the order of the slicing/mirroring operation, slice then mirror, just as a crop is performed before a mirror. The slice is non-axis aligned, yet mirroring is well defined due to the retention of the outside piece. The resulting sliced mesh is mirrored and has unique regions numbers in the internal mirrored segments also.

Syntax:

```
Specifymaskpoly maskname="m1" \\
P1="6.81, 5.55" \\
P2="5.69, 5.91" \\
P3="5.69, 7.09" \\
P4="6.81, 7.45" \\
P5="7.5, 6.5"
```

```
export atlas(conformal) \\
structure="conformal_slice_mask_mirror_x.str" \\
slice.mask="m1" \\
mirror="+x"
```

**Conclusions**

The export cropping and slicing operations provide a further level of control over the final structure exported from Victory Process cell mode. A standard cell mode export will retain the entire simulated structure. The cropping operation allows the extraction of a sub-section of this structure, either as a smaller cuboid (box crop), or a convex polygon (mask crop). This generalizes the cell mode exports to dump any output shape.

The slicing operation is similar to cropping, yet the part of the structure outside the slicing box or mask is also kept. The part outside is relabeled with unique region numbers to the part inside, and the entire structure is still dumped by the export. This allows for simple visualization of the internal parts of a structure in TonyPlot3D. In addition, multiple regions can now exist within the same material, allowing different properties to be set for different regions in a device simulator.
Defining VWF Curve Target For Curve Calibration

VWF allows a curve as target for optimization. This feature greatly benefits the users who need to calibrate TCAD simulation against measured curve such as SIMS. VWF offers multiple ways to use experimental curve as the target of optimization. In this article, two ways of defining curve target are discussed: 1) using vector target; 2) using target definition language of Dbinternal. Which method to choose depends on the optimizer that is being used to solve the problem. Method (1) is suitable for Levenberg-Marquardt, whereas method (2) needs to be used if any of the global optimizers (e.g., genetic optimization) is being selected.

Using Vector Target

Vector target uses the Y vector in a two-column matrix (X, Y). The matrix is formed in the format of TonyPlot user data file as shown in Figure 1. This file can be directly loaded into VWF (Figure 2), and only its Y values are used by VWF as the target for optimization. Once loaded the Y values are allowed to be manually modified. The vector target method requires the Levenberg-Marquardt optimization algorithm to be chosen. During curve calibration process, the Y column values in extracted curve from simulations will be compared to the Y values stored in VWF, an error vector is computed and passed to the Levenberg-Marquardt optimizer. The optimizer minimizes the difference between values of simulated Y and measured Y to achieve the purpose of calibration.

In order to make a meaningful comparison between calculated Ys (s denotes simulated values) and measured Ym (m denotes measured values), the two vectors must be ensured to have the same length and each Y element yi will be sampled at the same xi element from X vector. In reality, it can happen that the measured and simulated curves differ both in dimension as well as in their X values. In our method, the measured curve is therefore re-sampled to form a new Ym based on the Xs vector from extracted curve using numerical interpolation. For example, a measured data file (red curve in Figure 3) contains 92 entries, and the extracted green curve from TCAD simulation has 15 entries. The size of original measured curve needs to be reduced to 15 rows, and values of Y element are then obtained by interpolating the original measured data using the X vector from extracted curve (green curve). A script has been written for this purpose, and reduced size measured curve can be easily generated as shown in Figure 4. With newly generated vector target correct comparison between simulated data and measured data is guaranteed. The same script can also be used to add more data points to the original measurement file to match the data length in extracted curve from TCAD simulation.

**Example of TonyPlot User Data File**

<table>
<thead>
<tr>
<th>Boron vs Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>92 2 2</td>
</tr>
<tr>
<td>Boron</td>
</tr>
<tr>
<td>0.011 1.34e+10</td>
</tr>
<tr>
<td>0.0221 1.41e+10</td>
</tr>
<tr>
<td>0.0371 1.49e+10</td>
</tr>
<tr>
<td>0.04721 1.54e+10</td>
</tr>
<tr>
<td>0.0571 1.59e+10</td>
</tr>
<tr>
<td>0.0721 1.63e+10</td>
</tr>
<tr>
<td>0.08471 1.66e+10</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Figure 1. Example of TonyPlot user data file.

Figure 2. Loading vector target file.

Figure 3. Example of measured and simulated curves.
Using Target Definition Language of Dbinternal

VWF can use Dbinternal simulator to compute a scalar target from a given vector target. With this method the error computation for optimization will be transferred from VWF to Dbinternal simulator. The scalar target allows to use all available optimization algorithms in VWF for calibration. Users can also define their own error evaluation formula.

The error computation is performed using evalmath language of Dbinternal. In Figure 5 a short example is demonstrated. In the example two curves are read into two variables, and the variables are used for evaluating the error between two curves. The file Extracted.dat stores the data from simulation, and file Sims.dat contains the data from measurement. Two files must contain the same X vector in order to have a meaningful error evaluation in the subsequent steps. Therefore, the interpolation method described above may be applied to the measurement data to make sure measurement data participated in error evaluation are sampled at the same x points as those for extracted curve. An extract statement is needed to make calculated error target available for VWF.

Once the target evaluation block is imported to VWF, by right-clicking mouse in front of extract statement, a small window pops up to allow a selection of “error_target” to be an optimization target (Figure 6 and 7). For calibration purpose the target evaluation block should be combined with TCAD deck so that the extracted data will be updated automatically according to the parameter changes in the simulation.

Conclusion

In this article we outlined how a calibration task is run in VWF. We demonstrated the usage of different optimization strategies, Levenberg-Marquardt and global optimization, to calibrate model parameters to reproduce SIMS data. Interpolation was used to sample the measured data at the simulated depths. This setup allows for a fully automated calibration without user interaction.
JOIN THE WINNING TCAD TEAM

USA Headquarters:

Silvaco, Inc.
4701 Patrick Henry Drive, Bldg. 2
Santa Clara, CA 95054 USA

Phone: 408-567-1000
Fax: 408-496-6080

sales@silvaco.com
www.silvaco.com

Worldwide Offices:

Silvaco Japan
jpsales@silvaco.com

Silvaco Korea
krsales@silvaco.com

Silvaco Taiwan
twsales@silvaco.com

Silvaco Singapore
sgsales@silvaco.com

Silvaco Europe
eusales@silvaco.com