SOLSTICE ABG
(Absolute Beginner's Guide)

April 27, 2018

Méso-Star (https://www.meso-star.com)

Contents

1 Introduction

Solstice is a scientific software that computes the power collected by a concentrated solar power plant. The basic idea is that the user can provide the full geometry of the plant that will be used in the numerical simulation: the shape, position and spectral properties of each reflector must be provided, but the user can also provide a description of additional structural elements, such as mounting systems, through the definition of materials. The most basic solar plant should consist of at least one primary reflector and one receiver, but there is actually no limit on the number of reflectors (primary or not) and receivers that can be specified.

Solstice uses the Monte-Carlo method: the full trajectory of a given number of solar rays is randomly sampled, and the results of this statistical sampling is used to provide various results and the statistical uncertainty over each one of these results. **Solstice** performs only the direct computation of the concentrated solar power, for a given solar plant geometry. It does not to perform a automatic optimisation of the solar plant's design with respect to a given criteria, even if the direct computation can provide some insight for design purposes.

The code will not only evaluate the total concentrated power for each specified receiver, but also various losses: losses due to the cosine effect, shadowing of primary reflectors, absorption by materials (via imperfect reflectors or semi-transparent solids) and by the atmosphere. These losses are computed (with a statistical uncertainty) for each receiver, and for each primary reflector.

In terms of user interface, **solstice** is used as any Linux system command: from the command line. Input can be specified via various files and command-line options. Output is provided in the terminal, but as for any command-line tool, it can be written into output files for subsequent use.

This guide is aiming at providing a minimal support for the complete beginner. However, some knowledge about using a Linux system and a terminal is required. We also assume **solstice** was successfully installed and is ready to use. This guide was written for version 0.7.1 of **solstice**; however, input and output formatting should be very similar for subsequent versions. From version 0.4.0, installation is very easy since the pre-compiled package can be downloaded; the installation can be checked by issuing the "solstice -h" command.

One final but important note is that the user should not expect to obtain numerical values exactly similar to the various numerical results that are provided for the tests that are mentioned in this document. **solstice** results are expected to be different on different machines/systems due to different random number sequences. However, results should not differ by much, and should at least be within the range defined by numerical uncertainties.

2 First steps: performing a very simple computation

In this section, the user will be guided through the various steps required for the most basic computations. This will be a good excuse to introduce the various concepts used in **solstice**.

First, open a terminal. You will then have to set some environment variables in order to use the required version of **solstice**: in order to do that, you have to navigate to the /etc directory of the **solstice** version that you want to use, and then source the ”solstice.profile” file; for instance:

```
cd Solstice-0.7.1-GNU-Linux64/etc
source ./solstice.profile
```

You can check that the relevant environment variables have been set accordingly:

```
echo $PATH
```

should return something similar to:

```
/home/starlord/solstice/Solstice-0.7.1-GNU-Linux64/bin:/usr/local/sbin:/usr/local/bin:/
    usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

What matters is that the ”Solstice-0.7.1-GNU-Linux64/bin” directory is listed in this variable; similarly for the LD_LIBRARY_PATH variable:

```
echo $LD_LIBRARY_PATH
```

should look like:

```
/home/starlord/solstice/Solstice-0.7.1-GNU-Linux64/lib:
```

The ”Solstice-0.7.1-GNU-Linux64/lib” directory has to be listed.

This step (sourcing the ”solstice.profile” file) must be repeated every time a terminal window is opened for using **solstice**. You can check everything works as intended by running the ”solstice –version” command (no matter the directory you run this command into, since the $PATH variable should now be set), and the result should be consistent with the **solstice** version you chose to use.

You can now run the following command:

```
solstice -h
```

This should display the basic help about the **solstice** command.

Additional information can be found in the man pages for **solstice**; the following command:

```
man solstice
```

will display the main man page for **solstice**; as mentioned at the very end of this man page, additional man pages are available in order to describe the input syntax (man solstice-input), the output format (man solstice-output) and the syntax used to declare receivers (man solstice-receivers).

The purpose of this guide is to provide enough examples so that the inexperienced user may be able to explore **solstice** man pages, that are much more complete than the present document.

Instead of describing in more detail the list of possible **solstice** options, we are going to focus on the following example command:

```
solstice -D 0,90 -R receiver.yaml geometry.yaml
```

Now, using the output of the **solstice** help command, it is easy to see that:

- the ”-D” option is used to specify a main solar direction. The two subsequent numerical values are the azimuthal and zenithal directions (in degrees, see figure 1).
- the ”-R” option tells **solstice** that the receiver is defined in the ”receiver.yaml” input file.
- the ”geometry.yaml” file is the input file where the geometry of the solar plant is provided; no command-line option is necessary to specify this is a geometry file.

In this example, two input files are used; they contain information that uses the YAML specifications. In this example, these two files contain the following information:

The ”geometry.yaml” file:

```
- sun:
    dni: 1

- entity:
    name: reflector
    primary: 1
    transform:
      rotation: [-45, 0, 0]
      translation: [0, 0, 0]
    geometry:
    - material:
        back:
          mirror: {reflectivity: 1.0, slope_error: 0}
        front:
          mirror: {reflectivity: 1.0, slope_error: 0}
      plane:
        clip:
        - operation: AND
          vertices:
          - [-0.5,-0.5]
          - [-0.5, 0.5]
          - [ 0.5, 0.5]
          - [ 0.5,-0.5]

- entity:
    name: target
    primary: 0
    transform:
      rotation: [90, 0, 0]
      translation: [0, 10, 0]
    geometry:
    - material:
        back:
          matte: {reflectivity: 0}
        front:
          matte: {reflectivity: 0}
      plane:
        slices: 20
        clip:
        - operation: AND
          vertices:
          - [-5.0, -5.0]
          - [-5.0, 5.0]
          - [5.0, 5.0]
          - [5.0, -5.0]
```

And the "receiver.yaml" file:

```
- name: target
  side: FRONT_AND_BACK
```

These files provide an example of how the definition of the solar plant is structured. As shown in the geometry file, several elements are declared: the sun (with label "sun") and two entities ("reflector" and "target"). Finally, within the "receiver.yaml" file, the receiver is defined as the "target" entity. Let us give some more details about these two files:

- The only mandatory parameter for the definition of the sun is its DNI. The spectral distribution of energy can be specified (further examples will show how). Also, a model of the solar disk can be specified, as mentioned in the solstice-input man page: both a pillbox model and the Buie model are available. When no specific sun disk model is specified, all incoming solar radiation comes from the provided main solar direction (collimated radiation).

- The "reflector" entity is defined as a horizontal unit square whose center is (0,0). It is defined from the clipping of the z=0 plane: the AND clipping operation means only the square is retained. Alternatively, the clipping operation could be SUB, which would result in a square hole. When using planes, the default is to perform various operations on the horizontal (z=0) plane. Then the resulting shapes can be translated and rotated over each axis, which is the case in this example: the square is rotated by 45° around the X-axis. The "reflector" entity uses a material whose properties are defined for each face: in this example, both faces have a total reflectivity and a null slope_error (perfect specular mirror). Finally, the "reflector" entity is defined as a primary reflector: primary geometries are the ones that receive solar radiation, and thus constitute the beginning of each optical path.

- Similarly, the "target" geometric template defines a initially horizontal square of length 10, whose center is (0,0). This square is later both rotated by 90° around the X-axis and translated to the (x=0, y=10, z=0) position. The material used in the definition of this template is black (null reflectivity) on both faces. This entity is subsequently defined as the main receiver within the "receiver.yaml" file.

The "solar plant" that is described in these two files consists therefore in a 1 square meter perfect reflector plane tilted at 45°, that redirects the incoming solar radiation towards a vertical black bigger square receiver. All incoming sun rays come from the zenith (90° above the horizon) will hit the reflector. We can expect that, since the DNI is 1 W/m^2 and the projected surface of the reflector along the incoming sunlight direction is $1/\sqrt{2}\ m^2$, there is $1/\sqrt{2}$ watt of incoming radiative power at the surface of the reflector. Since it is a perfect mirror, all rays should bounce back along the horizontal and hit the receptor. The receptor will be hit by $1/\sqrt{2}$ watt of power on its front face: the front face of the initial (horizontal) square is its upper face. When this square is rotated by 90° around the X-axis, its upper face is now facing the reflector.

Let us make the computation using **solstice** and check it provides the expected results:

```
solstice -D 0,90 -R receiver.yaml geometry.yaml
```

This command should provide the following output:

```
1  #--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
2  7 1 1 10000 0
3  1 0
4  0.707107 1.22878e-09
5  0.707107 1.22878e-09
6  0 0
7  0 0
8  0 0
9  0 0
10 target 2 100   0.707107 1.22878e-09   0.707107 1.22878e-09   0.707107 1.22878e-09   0 0
         0 0   0.707107 1.22878e-09   0.707107 1.22878e-09   0.707107 1.22878e-09   0 0
       0 0   0.707107 1.22878e-09   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0
       0 0   0 0
11 reflector 6 1 10000   0.707107 1.22878e-09   0 0
12 2 6   0.707107 1.22878e-09   0.707107 1.22878e-09   0.707107 1.22878e-09   0 0   0 0
     0.707107 1.22878e-09   0.707107 1.22878e-09   0.707107 1.22878e-09   0 0   0 0   0
     0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0
```

As indicated in the solstice-output man-page, this output must be read as follows:

- The first line recalls the provided sun direction. This is a particular case, since we provided only one incoming sun direction: we could have provided as many as required, and the whole above output information set would have been produced for each sun direction.
- On the second line, the 7 indicates how many global results have been computed (lines 3 to 9). Then is recalls the number of receivers (only 1 in our case), the number of primary reflectors (also 1), the default number of statistical realisations (10000) since we did not provide it explicitly, and the number of realisations that have failed due to numerical precision issues (0).
- The next seven lines (3 to 9) provide the numerical values of the global results:
  1. the potential flux (W) incoming on all receivers if they were optimally oriented (in this case, the receiver would be optimally oriented if it had been left horizontal). Two values are provided: the potential flux (1 W) and its numerical uncertainty (0 W). The result should then be read as 1 W $\pm$ 0 W.
  2. the flux absorbed by all receivers: 0.707107 W $\pm$ 0 W.
  3. the cos factor: 0.707107 $\pm$ 0; it is defined as the average (over the surface of all primary reflectors) cosinus of the angle between the incoming solar radiation and the normal of primary reflectors.
  4. shadow losses: 0 W $\pm$ 0 W; this is the total flux that does not reach primary reflectors due to shadowing of these reflectors. In our case, there is none.
  5. missing losses: 0 W $\pm$ 0 W; this is the flux that reached primary reflectors, is reflected, not absorbed by any solid (when reflected or transmitted) or by the atmosphere, but is not finally absorbed by receivers. In this very simple first example, no path misses the receiver.

6. reflectivity losses: 0 W ± 0 W; this is the flux that has been absorbed by elements other that receivers (both solid surfaces and solid volumes such as semi-transparent materials).
7. absorptivity losses: 0 W ± 0 W; this is the flux that is absorbed by the atmosphere on the path from a primary reflector to a receiver. Since we did not provide any information about atmospheric absorptivity, this value is null.

- Then one line of results is provided for each receiver (in this case, only one: line number 10). It provides the name of the receiver, its ID, its area in square meters, then 11x2 sets of results (result and its numerical uncertainty) for each side of the receiver (front and back). These 11 sets of results, for each side, are the following:

  1. incoming flux over the receiver's side
  2. incoming flux over the receiver's side if no absorption occurs in materials
  3. incoming flux over the receiver's side if no extinction occurs in the atmosphere
  4. flux removed because of absorption in materials
  5. flux removed because of extinction in the atmosphere
  6. the flux that was absorbed by the receiver's side
  7. the flux that would be absorbed by the receiver's side if no absorption occurs in materials
  8. the flux that would be absorbed by the receiver's side if no extinction occurs in the atmosphere
  9. flux that has not been absorbed by the receiver's side because of absorption by materials
  10. flux that has not been absorbed by the receiver's side because of extinction by the atmosphere
  11. global efficiency: the fraction of the potential flux that was absorbed by the receiver's side

  In this case, these results are null (0 ± 0 watts) for the back side since radiation is absorbed once it reaches the front side: as expected, a total of 0.707107 watt reached the front side of the receiver, and all of it was absorbed because of the perfectly absorbing receiver, and nothing was lost because of absorption by imperfect reflectors, semi-transparent materials or the atmosphere.

- One line is then provided for each primary entity (one in our case: line number 11); it provides the name of the primary reflector, its ID, its area in square meters, the number of statistical realisations that have been sampled on this reflector, and two sets of results for this primary reflector:

  1. the cos-factor of the reflector (0.707107 ± 0): average, over the surface of the reflector, of the cosinus of the angle between the direction of incoming radiation and the local normal to the surface.
  2. shadow losses of the reflector (0 ± 0): flux that did not reach the reflector because of primary reflectors shadowing.

- Finally, one line is provided for each (receiver / primary reflector) couple. In our case, since there are only one receiver and one primary reflector, there is only one such line (line 12). It provides the ID of the receiver, the ID of the primary reflector, and 10 sets of results per receiver's side (front / back):

1. incoming flux over the receiver's side, coming from the reflector
2. incoming flux over the receiver's side if no absorption occurs in materials, coming from the reflector
3. incoming flux over the receiver's side if no extinction occurs in the atmosphere, coming from the reflector
4. flux removed because of absorption in materials, coming from the reflector
5. flux removed because of extinction in the atmosphere, coming from the reflector
6. the flux that was absorbed by the receiver's side, coming from the reflector
7. the flux that would be absorbed by the receiver's side if no absorption occurs in materials, coming from the reflector
8. the flux that would be absorbed by the receiver's side if no extinction occurs in the atmosphere, coming from the reflector
9. flux that has not been absorbed by the receiver's side because of absorption by materials, coming from the reflector
10. flux that has not been absorbed by the receiver's side because of extinction by the atmosphere, coming from the reflector

In this case, only front results are non-null: as expected, all the reflected power (0.707107 watt) effectively reached the front side of the receiver, and all of it was absorbed. Nothing was absorbed by solid surfaces or semi-transparent materials, and nothing was absorbed by the atmosphere.

Solstice can be used in order to generate a 3D model of the geometry, using the "-g" flag. Try using the following command:

```
solstice -n 100 -g format=obj -t1 -D 0,90 -R receiver.yaml geometry.yaml > geom.obj
```

This will produce the "geom.obj" file that can subsequently be visualised by any software that can render .obj files (such as *meshlab*). Figure 2 provides a visualisation of the corresponding 3D model.

Finally, **solstice** can produce a visualisation of a given number of sunlight ray paths (for instance 100), using the following command:

```
solstice -n 100 -p default -t1 -D 0,90 -R receiver.yaml geometry.yaml | sed '1d' > rays
    .vtk
```

This produces the "rays.vtk" output file, that can be used by *paraview* (together with the previously produced "geom.obj" file) in order to produce figure 3. In this second figure, both the reflector and the receiver can be seen, along with the required 100 light paths (randomly sampled over the primary reflector): since the incoming and the reflection directions are identical, they can not be distinguished on this figure. However, blue paths are the ones that effectively reach a receiver.

3 Tweaking the first example

We will now experiment small variations of the example provided in section 2 in order to learn how to solve more complex problems.

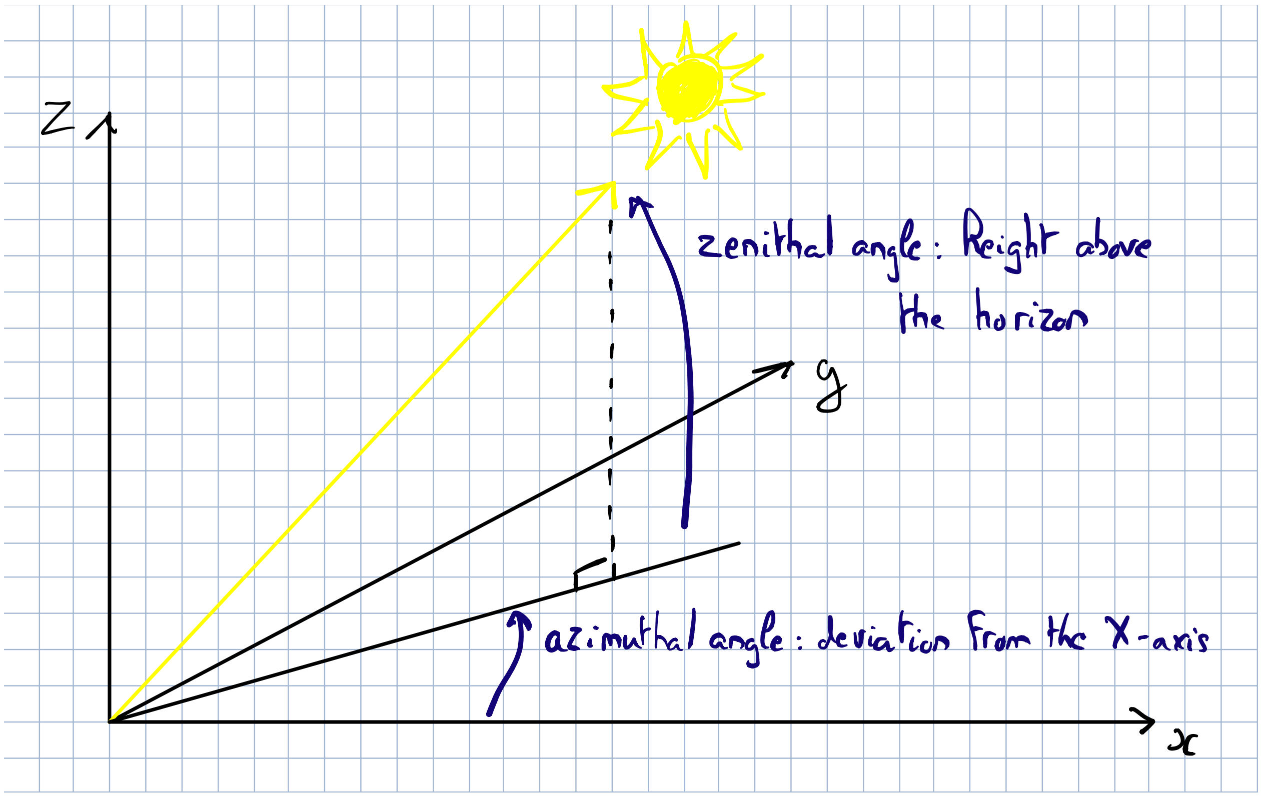
Z
zenithal angle: height above the horizon
y
azimuthal angle: deviation from the X-axis
x

Figure 1: Definition of the zenithal and azimuthal angles

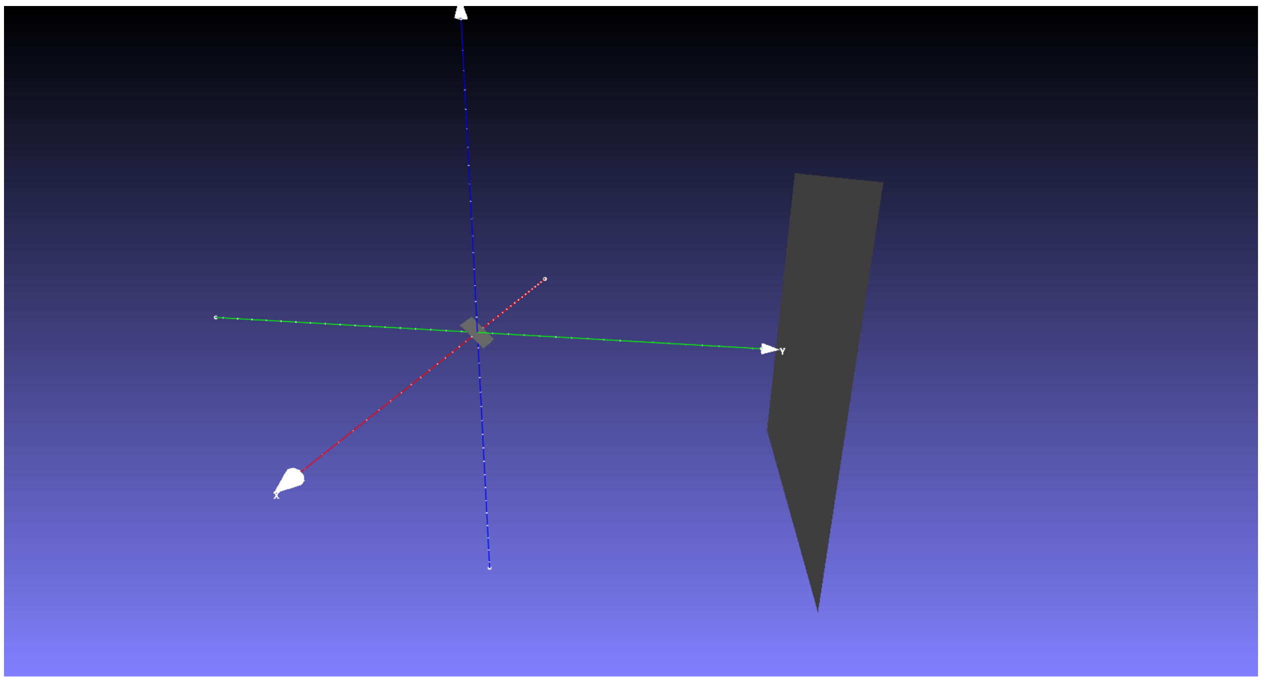

Figure 2: 3D model of the geometry used in the first example

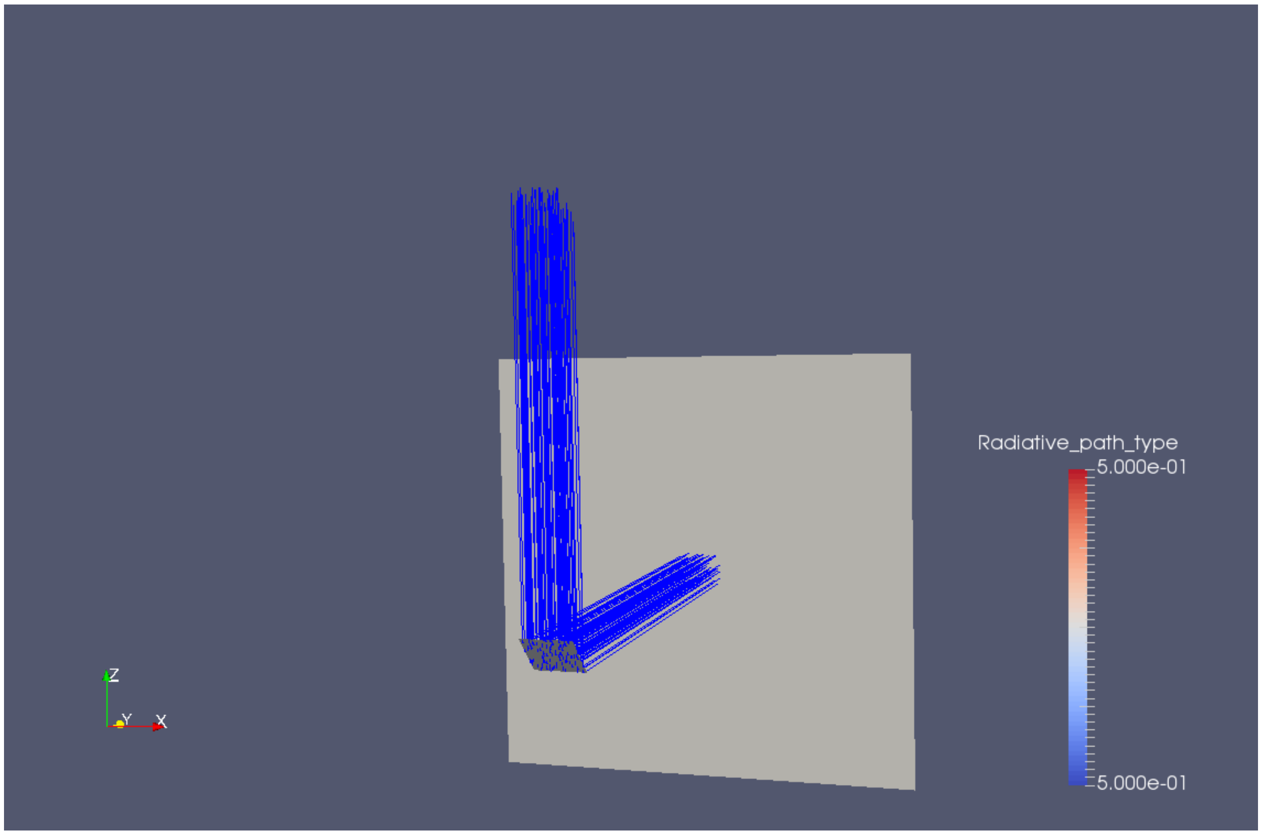

Figure 3: 3D visualisation of 100 ray paths in the first example

3.1 Solar disk models

The default model of the solar disk is that there is no solar disk: all incoming sunlight comes from the specified sun direction. You can alternatively use two different solar disk models: the pillbox model and the Buie model.

Try replacing the ”sun” section in the ”geometry.yaml” file by the following lines:

```
- sun: &sun
    dni: 1
    pillbox:
      half_angle: 0.53
    spectrum: [{wavelength: 1, data: 1}]
```

In this example, we tell **solstice** to use the *pillbox* model with an solar disk half-angle of 0.53°: the solar disk is now considered as a surface located at the end of a cone of 0.53° half-angle, with a uniform flux. The ”spectrum” attribute is optional, it could be removed in this example. Section 4 will describe how to use more complex spectra.

Then run the **solstice** computation again, with the same main solar direction:

```
solstice -D 0,90 -R receiver.yaml geometry.yaml
```

Produces:

```
#--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
7 1 1 10000 0
1 0
0.707075 3.27624e-05
0.707107 1.22878e-09
0 0
0 0
0 0
0 0
target 2 100   0.707075 3.27624e-05   0.707075 3.27624e-05   0.707075 3.27624e-05   0 0
      0 0   0.707075 3.27624e-05   0.707075 3.27624e-05   0.707075 3.27624e-05   0 0
     0 0   0.707075 3.27624e-05   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0
     0 0   0 0
reflector 6 1 10000   0.707107 1.22878e-09   0 0
2 6   0.707075 3.27624e-05   0.707075 3.27624e-05   0.707075 3.27624e-05   0 0   0 0
   0.707075 3.27624e-05   0.707075 3.27624e-05   0.707075 3.27624e-05   0 0   0 0   0
   0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0
```

The main difference with the first set of results is that the computed power (incoming on the reflector, absorbed by the front face of the receiver) is computed with a non-null standard deviation. The direction of incoming solar radiation is now sampled within a small solid angle centered on the main solar direction.

We can also use the Buie model (presented in [1]). For instance, try replacing the ”sun” section in the ”geometry.yaml” file by the following lines:

```
- sun: &sun
    dni: 1
    buie:
      csr: 0.1
    spectrum: [{wavelength: 1, data: 1}]
```

We have now specified the use of the Buie model, using a circumsolar ratio of 0.1; the distribution of solar flux varies with the position of emission along the solar disk, and a circumsolar ring due to atmospheric scattering is also taken into consideration. The output of **solstice** in this case is the following:

```
#--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
7 1 1 10000 0
1 0
0.707053 3.14633e-05
0.707107 1.22878e-09
0 0
0 0
0 0
0 0
target 2 100   0.707053 3.14633e-05   0.707053 3.14633e-05   0.707053 3.14633e-05   0 0
     0 0   0.707053 3.14633e-05   0.707053 3.14633e-05   0.707053 3.14633e-05   0 0
    0 0   0.707053 3.14633e-05   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0
    0 0   0 0
reflector 6 1 10000   0.707107 1.22878e-09   0 0
2 6   0.707053 3.14633e-05   0.707053 3.14633e-05   0.707053 3.14633e-05   0 0   0 0
   0.707053 3.14633e-05   0.707053 3.14633e-05   0.707053 3.14633e-05   0 0   0 0   0
   0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0   0 0
```

Once again, the dispersion of incoming sunlight directions translates only in a modification of the cos-factor and the total power that is absorbed by the receiver.

3.2 Surface properties of materials

In the first example, the definition of the material was embedded in the definition of the reflector. Is is possible to define surface properties independently of entities. For instance, let us create a "material" section in the "geometry.yaml" file, prior to the definition of entities:

```
- material: &custom
    mirror: { reflectivity: 0.75, slope_error: 0 }
```

The primary reflector must also be modified in order to use the "custom" material:

```
- entity:
    name: reflector
    primary: 1
    transform:
      rotation: [-45, 0, 0]
      translation: [0, 0, 0]
    geometry:
    - material: *custom
      plane:
        clip:
        - operation: AND
          vertices:
          - [-0.5,-0.5]
          - [-0.5, 0.5]
          - [ 0.5, 0.5]
          - [ 0.5,-0.5]
```

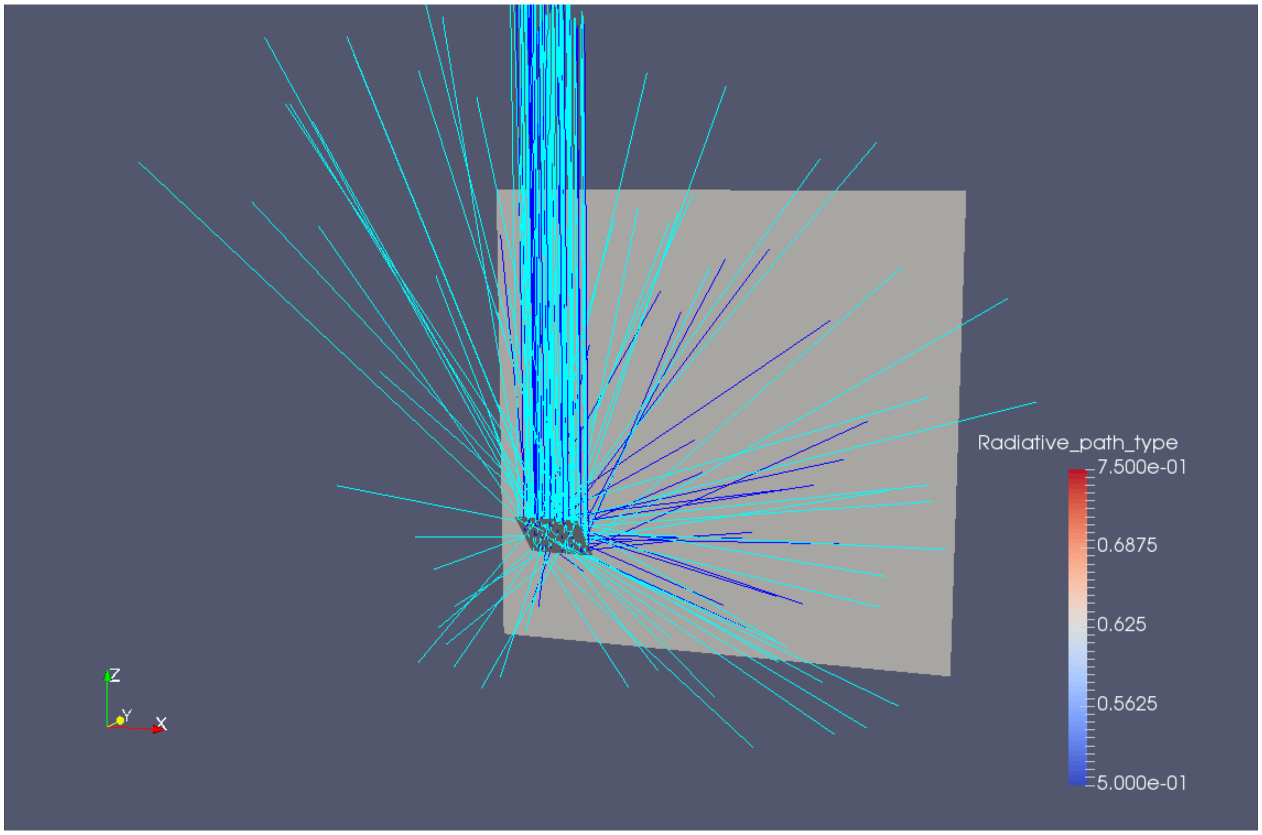

Figure 4: 3D visualisation of 100 ray paths in the example of a custom material with reflectivity=0.75 and slope_error=0.5

This mechanism provides the possibility to define the surface properties of as many materials as required, and to easily switch materials within the definition of the various entities.

The output of the **solstice** computation is now:

```
#--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
7 1 1 10000 0
1 0
0.53033 2.11523e-09
0.707107 1.22878e-09
0 0
0 0
0.176777 3.07195e-10
0 0
target 2 100    0.53033 2.11523e-09    0.707107 1.22878e-09    0.53033 2.11523e-09
    0.176777 3.08323e-10    0 0    0.53033 2.11523e-09    0.707107 1.22878e-09    0.53033
    2.11523e-09    0.176777 3.08323e-10    0 0    0.53033 2.11523e-09    0 0    0 0    0 0
    0 0    0 0    0 0    0 0    0 0    0 0    0 0    0 0
reflector 6 1 10000    0.707107 1.22878e-09    0 0
2 6    0.53033 2.11523e-09    0.707107 1.22878e-09    0.53033 2.11523e-09    0.176777
    3.08323e-10    0 0    0.53033 2.11523e-09    0.707107 1.22878e-09    0.53033 2.11523e
    -09    0.176777 3.08323e-10    0 0    0 0    0 0    0 0    0 0    0 0    0 0    0 0    0 0
    0 0    0 0
```

Using a reflectivity of 0.75 instead of 1 has no consequence of the cos-factor, but the power that is absorbed by the receiver is now 0.53033±0 watts. We can also see that 0.176777±0 watts have been absorbed by the reflector due to its partial reflectivity (within the global results, in the results line for the receiver and the results line for the receiver/reflector couple).

We can also modify the value of the "slope_error" parameter in the properties of our "custom" material; let us give it a non-null value:

```
- material: &custom
      mirror: { reflectivity: 0.75, slope_error: 0.5 }
```

This "slope_error" parameter controls the reflection BRDF. We can see a clear effect on computation results:

```
#--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
7 1 1 10000 0
1 0
0.096772 0.00204819
0.707107 1.22878e-09
0 0
0.399882 0.00196913
0.210453 0.000577128
0 0
target 2 100    0.096772 0.00204819    0.129047 0.00273124    0.096772 0.00204819
    0.032275 0.00068322    0 0    0.096772 0.00204819    0.129047 0.00273124    0.096772
    0.00204819    0.032275 0.00068322    0 0    0.096772 0.00204819    0 0    0 0    0 0    0
    0    0 0    0 0    0 0    0 0    0 0    0 0    0 0
reflector 6 1 10000    0.707107 1.22878e-09    0 0
2 6    0.096772 0.00204819    0.129047 0.00273124    0.096772 0.00204819    0.032275
    0.00068322    0 0    0.096772 0.00204819    0.129047 0.00273124    0.096772 0.00204819
      0.032275 0.00068322    0 0    0 0    0 0    0 0    0 0    0 0    0 0    0 0    0 0    0 0
      0 0
```

The power that reaches the receiver's back face is now significantly lower than the initial 0.707107 watts, and now the flux lost to the environment (missing-loss) is non-null. Also, the additional flux that could have been absorbed by the receiver if the mirror was a perfect reflector is 3.25446e-2 ± 6.86322e-4 watts. As can be seen in figure 4, reflection directions are scattered over the upper hemisphere, and some rays are not reflected in the direction of the receiver (turquoise rays).

3.3 Rotating and translating geometric elements

As can be seen in the "geometry.yaml" file, a "transform" section can be added to the definition of any given entity in order to apply a rotation of an arbitrary angle around each axis, and also to apply a arbitrary translation along each axis. Positioning geometries in space is pretty straightforward from the provided example.

3.4 Geometric shapes

As mentioned in the solstice-input man page, the following geometric shapes can be defined in **solstice**: cuboids, cylinders, spheres, hemispheres, planes, paraboloids, hyperboloids, parabolic cylinders. There is also the possibility to include geometries that are defined using the STL format. Furthermore, clipping operations can be performed over parametric shapes (hemispheres, paraboloids, hyperboloids, parabolic cylinders and planes): this was the case throughout all previous examples, when defining the two squares from a original plane.

The user should refer to the man pages that provide extensive documentation on the way these shapes can be defined. We will use various shapes in the following examples.

3.5 Pivots

Pivots provide the possibility to automatically rotate a given geometry in order to reflect incoming sunlight toward a given position. In **solstice**, two types of pivots are defined: the *x_pivot* and the *zx_pivot*. A example showing how to use both types will be provided.

The *x_pivot* makes possible a rotation of a geometry around the X-axis only. It is easy to define any direction as the rotation axis of the pivot by first applying a rotation to the pivot. Let us take the example of the following "self_oriented_parabol.yaml" file:

```
- sun: &sun
    dni: 1
    spectrum: [{wavelength: 1, data: 1}]

- material: &specular
      mirror: { reflectivity: 1, slope_error: 0 }

- material: &black
      matte: { reflectivity: 0 }

- template: &self_oriented_parabol
    name: "so_parabol"
    transform: { translation: [0, 0, 4], rotation: [0, 0, 90] }
    x_pivot:
      ref_point: [0, 0, 0]
      target: { sun: *sun }
    children:
      - name: "parabol"
        primary: 1
        geometry:
        - material: *specular
          parabol:
            focal: 4
            clip:
            - operation: AND
              vertices: [[-5.0, -5.0], [-5.0, 5.0],
               [5.0, 5.0], [5.0, -5.0]]
      - name: "small_square"
        transform: { translation: [0, 0, 4] }
        primary: 0
        geometry:
        - material:  *black
          plane:
            clip:
              - operation: AND
                vertices: [[-0.50, -0.50], [-0.50, 0.50],
                 [0.50, 0.50], [0.50, -0.50]]

- entity:
    name: "reflector"
    transform: { rotation: [0, 0, 0], translation: [0, 0, 0] }
    children: [ *self_oriented_parabol ]
```

In this file, the "self_oriented_parabol" geometric template is constructed first by defining a global transform over the whole template: it will be globally translated by the [0 0 4] vector and rotated by an angle of 90° around the Z-axis. The purpose of this rotation is to change the rotation axis of the pivot from the X-axis to the Y-axis. Then the *x_pivot* is defined as the first element of the template. Its target is defined as the sun: the result is that the normal of the paraboloid will be rotated (within the limits of a single-axis pivot) so that it matches with the sun direction. Then a children to the pivot is defined: its geometric shape is a paraboloid, that is clipped by a square of length 10 centered around the Z-axis. It uses the "specular" material that is defined at the top of the file, and is defined as a primary reflector. Then a square of length 1 is defined, with a initial translation of 4 along the Z-axis. It uses the "black" material defined at the top of the file, and is not a primary reflector. Finally, a "reflector" entity is defined using the "self_oriented_parabol" template.

In this example, here is how things work when **solstice** parses this file: the paraboloid is created with its initial axis of symmetry along the Z-axis. The "small_square" geometry will be located at an altitude of 4 in the local referential of the paraboloid: the center of this square is then located at the focal point of the paraboloid. Finally, the whole geometry is translated by the [0 0 4] vector and rotated by 90° around the Z-axis.

In order to perform the **solstice** computation, we also need the following "parabol_receiver.yaml" file:

```
- { name: "reflector.so_parabol.small_square", side: BACK }
```

This file provides the path of the geometry that is used as a receiver in the yaml tree of the geometry file, and it is specified that only the back side of this geometry is used to receive some power.

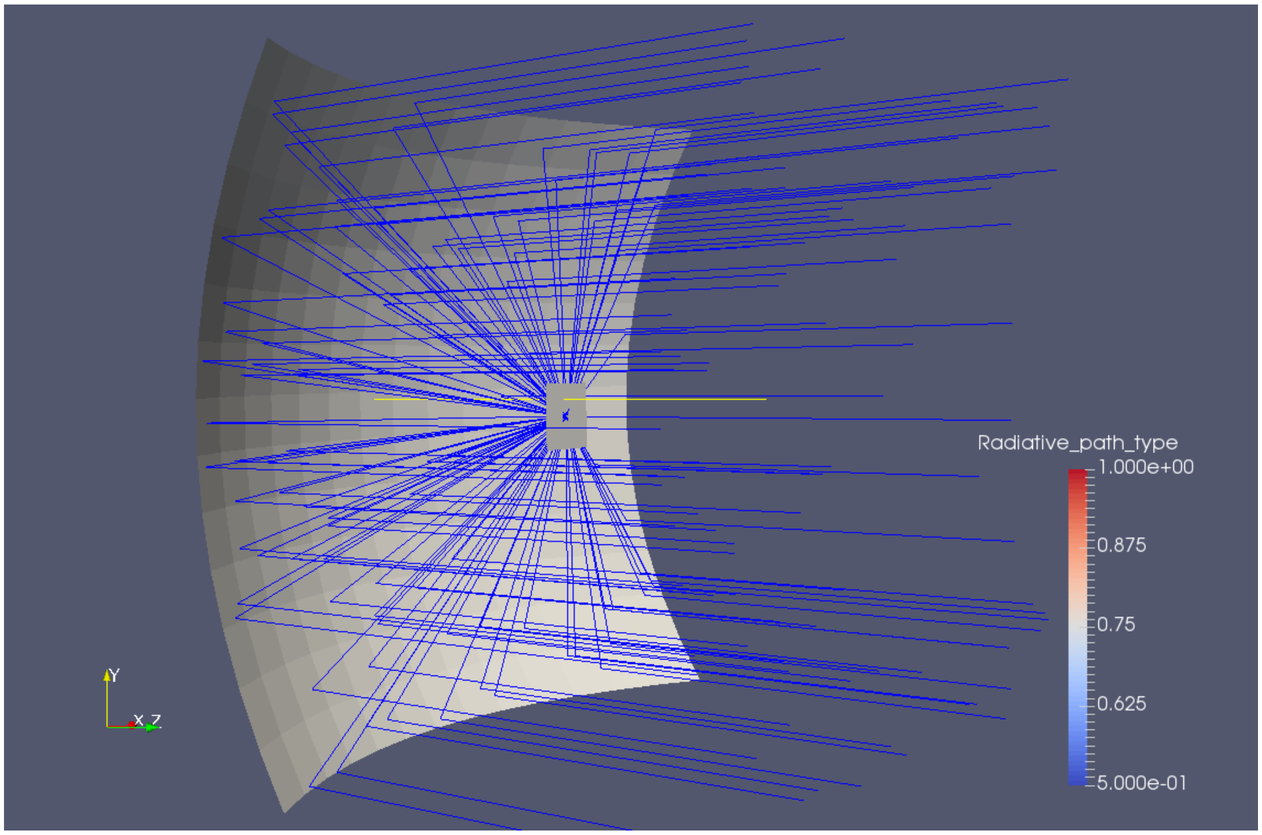

Figure 5: 3D visualisation of 100 ray paths in the example of a 45° sun direction, and a receiver that is located at the focal point of a self-oriented paraboloid pointing at the sun.

We can finally perform the computation:

```
solstice -D 0,45 -R parabol_receiver.yaml self_oriented_parabol.yaml
```

with the following result:

```
#--- Sun direction: 0 45 (-0.707107 -0 -0.707107)
7 1 1 10000 0
111.97 0
98.91 0.103833
0.893096 0
1.09 0.103833
0 0
0 0
0 0
reflector.so_parabol.small_square 6 1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1
      -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    98.91 0.103833    98.91 0.103833    98.91
    0.103833    0 0    0 0    98.91 0.103833    98.91 0.103833    98.91 0.103833    0 0    0 0
      0.883361 0.000927324
reflector.so_parabol.parabol 2 111.97 10000    0.893096 0    1.09 0.103833
6 2    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1    -1 -1
    98.91 0.103833    98.91 0.103833    98.91 0.103833    0 0    0 0    98.91 0.103833
    98.91 0.103833    98.91 0.103833    0 0    0 0
```

Over the 111.97±0 watts incoming on the surface of the paraboloid, only 98.81±0.108 watts are focused on the receiver: this is partly due to a cos-factor different from 1 (the potential flux is 111.97 watts because the DNI is 1 watt per square meter, and the surface of the paraboloid is 111.97 square meters; however, because of the curvature of the paraboloid, it intercepts only 100 watts: 111.97**cos-factor*), and to the fact that 1.19±0.108 watts are lost because of the shadow that the receiver casts on the paraboloid. No power is lost by the receiver and there are no reflectivity losses due to absorption. Figure 5 provides a visualisation of the scene.

The *zx_pivot* makes possible a rotation around two perpendicular axis. In the following "two_reflectors_one_receiver.yaml" file, two paraboloids are defined from the same "self_oriented_parabol" geometric template (a paraboloid with a focal distance of 100 meters). These two reflectors are located at different spatial directions, and reflect radiation back to a common square receiver.

```
- sun: &sun
    dni: 1
    spectrum: [{wavelength: 1, data: 1}]
# Possibility to provide as many wavelength values (and corresponding
# solar irradiance at ground level) as required, in order to
# take into account the spectral dependence to the solar emission
# and atmosphere absorption.

# Definition of materials
- material: &lambertian
    front:
      matte: { reflectivity: 1 }
    back:
      matte: { reflectivity: 1 }

- material: &specular
      mirror: { reflectivity: 1, slope_error: 0 }
```

```

- material: &black
      matte: { reflectivity: 0 }

# Definition of the "small_square" geometry that will be used
# by the receiver
- geometry: &small_square
    - material: *black
      plane:
        clip:
          - operation: AND
            vertices:
              - [-0.50, -0.50]
              - [-0.50, 0.50]
              - [0.50, 0.50]
              - [0.50, -0.50]

# Definition of the receiver's entity.
- entity:
    name: "square_receiver"
    primary: 0
# set "primary" to 0 because this entity should not the sampled
# as a primary reflector
    transform: { rotation: [0, 90, 0], translation: [100, 0, 10] }
# The "small_square" geometry is created as horizontal. In order to
# have it vertical, a rotation of 90 degrees around the Y-axis
# must be performed.
# Also, a translation is applied in order to position it in space.
    anchors:
      - name: "anchor0"
        position: [0, 0, 0]
# An anchor is defined, at a local position (in the referential of
# the square)
    geometry: *small_square


# Definition of a geometry template for primary reflectors (parabol)
- template: &self_oriented_parabol
    name: "so_parabol"
    transform: { translation: [0, 0, 4], rotation: [0, 0, 90] }
# The 90-degrees rotation around the Z-axis is required so that the
# pivot rotates around the Y-axis and not the X-axis.
    zx_pivot:
      ref_point: [0, 0, 0]
      target: { anchor: square_receiver.anchor0 }
# the target is the anchor that was previously defined in
# the receiver's entity (the center of the square)
    children:
      - name: "parabol"
        transform: { rotation: [-90, 0, 0], translation: [0, 0, 0] }
# This -90-degrees rotation over the X-axis is required because:
# - the parabol is generated "horizontal": its central axis is the Z-axis
# - the automatic orientation of the parabol will be performed so that
# the Y-axis is the local normal of the specular reflexion @ ref_point
# - a 90-degrees rotation around the Z-axis is applied
# => in order to make the X-axis the local normal of specular reflexion ,
# the parabol must be rotated by -90-degrees around X and then
# by 90-degrees around Z.
        primary: 1 # primary=1 -> sampled for integration
        geometry:
        - material: *specular
          parabol:
            focal: 100
            clip:
            - operation: AND
              vertices: [[-5.0, -5.0], [-5.0, 5.0], [5.0, 5.0], [5.0, -5.0]]

# Two identical parabols are placed at two different locations.
# Automatic orientation of the parabols so that a specular
# reflexion of the sun's main direction at the center of the parabol
# is directed to the center of the receiver.
- entity:
    name: "reflector1"
    transform: { rotation: [0, 0, 0], translation: [0, 0, 0] }
    children: [ *self_oriented_parabol ]

- entity:
    name: "reflector2"
    transform: { rotation: [0, 0, 0], translation: [10, 43.6, 0] }
    children: [ *self_oriented_parabol ]
```

In this example, the same geometric template is used to define both primary reflectors: this ”self_oriented_parabol” uses a *zx_pivot* in order to position each reflector so that radiation is sent back to the center of the receiver (using the notion of anchor).

The following ”common_receiver.yaml” file is used to define the receiver:

```
- { name: "square_receiver", side: BACK }
```

We can finally run **solstice**:

```
solstice -n 100000 -t1 -D 0,45 -R common_receiver.yaml two_reflectors_one_receiver.yaml
```

with the following result:

```
#--- Sun direction: 0 45 (-0.707107 -0 -0.707107)
7 1 2 100000 0
200.04 0
176.935 0.121085
0.925156 3.46816e-05
0 0
8.13302 0.119025
0 0
0 0
square_receiver 2 1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
    -1 -1   -1 -1   -1 -1   176.935 0.121085   176.935 0.121085   176.935 0.121085   0
    0   0 0   176.935 0.121085   176.935 0.121085   176.935 0.121085   0 0   0 0
    0.884499 0.000605303
reflector2.so_parabol.parabol 8 100.02 50222   0.915688 2.58316e-05   0 0
reflector1.so_parabol.parabol 6 100.02 49778   0.934708 2.29155e-05   0 0
```

```
2 8   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1
    83.8607 0.288723   83.8607 0.288723   83.8607 0.288723   0 0   0 0   83.8607
    0.288723   83.8607 0.288723   83.8607 0.288723   0 0   0 0
2 6   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1   −1 −1
    93.0744 0.295646   93.0744 0.295646   93.0744 0.295646   0 0   0 0   93.0744
    0.295646   93.0744 0.295646   93.0744 0.295646   0 0   0 0
```

Within the **solstice** command, the "-n 100000" option is used to explicitly specify the code will have to sample 10^5 statistical realisations (in order to achieve a higher numerical accuracy), and the "-t1" option tells the code to run the computation on 1 thread, even if by default it uses all available threads.

Similarly to the previous example, only a fraction of the radiation available on the reflectors is focused on the receiver. This is partly due to a cos-factor different from 1. And even if there is no shadow over the reflectors, there is some power that is reflected but does not reach the receptor (8.13±0.12 watts) because even if incoming sunlight is collimated, the sun direction does not match the normal of the second paraboloid ("reflector2").

Figure 6 provides a visualisation of the scene. We can see some turquoise paths that originate from the "reflector2" paraboloid.

4 A more complex example

The example provided in this section is not intended at describing a realistic solar plant. Instead, its purpose is to use as many concepts as possible in the definition of the geometry.

In the following "multiple_reflectors.yaml" file, three primary reflectors made from parabolic sections, with different focal distances, focus incoming sunlight at a common point. This point happens to be the upper focal point of a secondary reflector, in the shape of a hyperboloid (beam-down). Then a receptor is located at the second focal position of the hyperboloid. But in this last part of the optical path, a glass box has been placed. Figures 7 and 8 show the geometric configuration and optical paths when the box is set to use two different refractive materials.

```
# Definition of spectra:
# spectrum of incoming sunlignt
- spectrum: &solar_spectrum
  - {wavelength: 0.30, data: 1.0}
  - {wavelength: 0.40, data: 2.0}
  - {wavelength: 0.50, data: 0.5}
  - {wavelength: 0.60, data: 3.5}
  - {wavelength: 0.70, data: 1.5}
  - {wavelength: 0.80, data: 0.8}
# spectrum of air absorption coefficient
- spectrum: &air_kabs
  - {wavelength: 0.30, data: 1.0e-4}
  - {wavelength: 0.40, data: 1.0e-5}
  - {wavelength: 0.50, data: 2.0e-5}
  - {wavelength: 0.60, data: 2.0e-4}
  - {wavelength: 0.70, data: 3.0e-5}
  - {wavelength: 0.80, data: 1.0e-4}
# spectrum of glass absorption coefficient
- spectrum: &glass_kabs
```

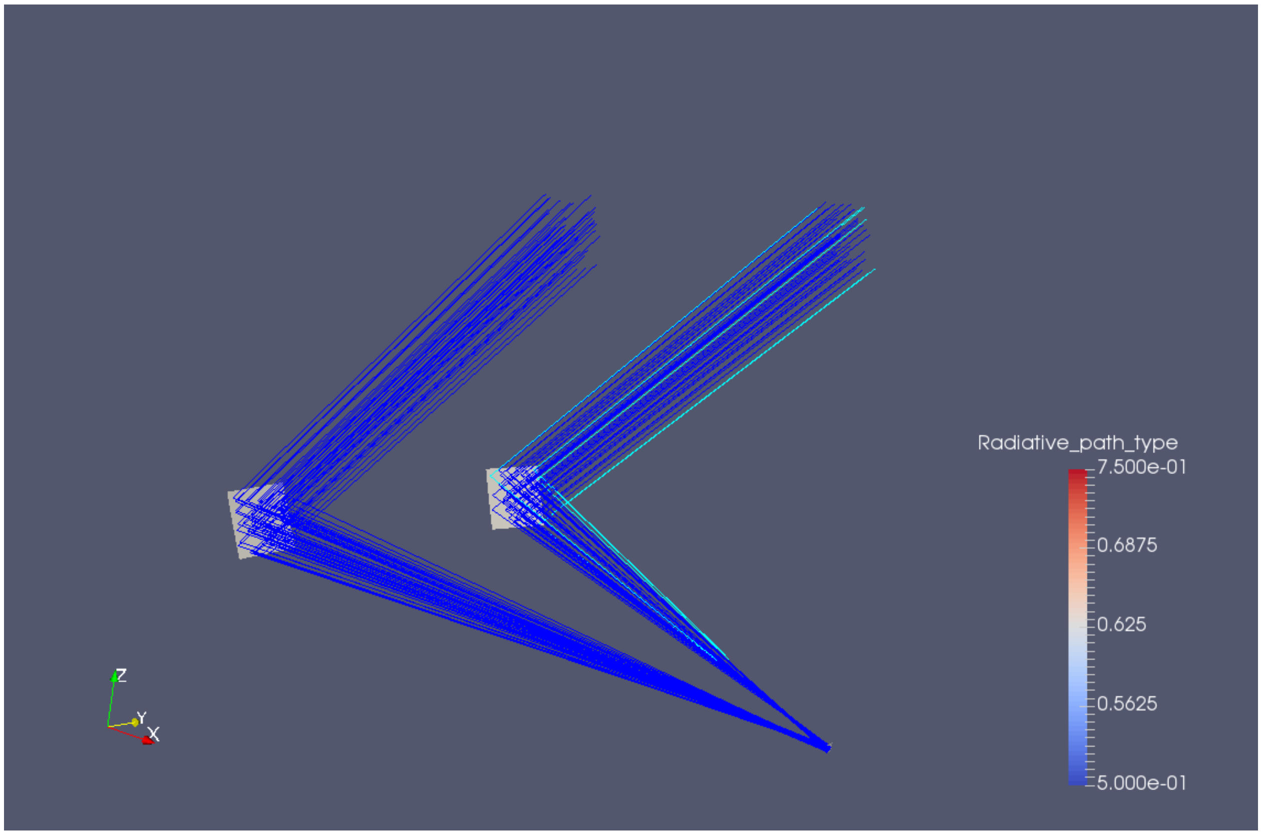

Figure 6: 3D visualisation of 100 ray paths in the example of a 45° sun direction, and a receiver that is located at the focal position of two paraboloids that are self-oriented using a *zx_pivot*.

```
  - {wavelength: 0.30, data: 1.0e-2}
  - {wavelength: 0.40, data: 1.0e-3}
  - {wavelength: 0.50, data: 2.0e-3}
  - {wavelength: 0.60, data: 2.0e-2}
  - {wavelength: 0.70, data: 3.0e-3}
  - {wavelength: 0.80, data: 1.0e-3}
# spectrum of glass refractive index
- spectrum: &glass_ref_index
  - {wavelength: 0.30, data: 1.40}
  - {wavelength: 0.40, data: 1.39}
  - {wavelength: 0.50, data: 1.37}
  - {wavelength: 0.60, data: 1.34}
  - {wavelength: 0.70, data: 1.30}
  - {wavelength: 0.80, data: 1.25}

- sun: &sun
    dni: 1
    buie:
      csr: 0.05
    spectrum: *solar_spectrum

# Definition of media
# medium: air
- medium: &air_medium
    refractive_index: 1
    extinction: *air_kabs
# medium: glass
- medium: &glass_medium
    refractive_index: *glass_ref_index
    extinction: *glass_kabs

# Definition of materials
# lambertian reflection
- material: &lambertian
    front:
      matte: { reflectivity: 1 }
    back:
      matte: { reflectivity: 1 }
# specular reflexion
- material: &specular
    mirror: { reflectivity: 1, slope_error: 0 }
# blackbody (no reflexion)
- material: &black
    mirror: { reflectivity: 0, slope_error: 0 }
# thin glass: uses the "thin_dielectric" material
- material: &thin_glass
    thin_dielectric:
      thickness: 0.001
      medium_i: *air_medium
      medium_t: *glass_medium
# glass: uses the "dielectric" material
- material: &glass
    front:
      dielectric:
        medium_i: *air_medium
        medium_t: *glass_medium
    back:
      dielectric:
        medium_i: *glass_medium
        medium_t: *air_medium

# Definition of the "small_square" geometry that will be used
# by the receptor
- geometry: &small_square
    - material: *black
      plane:
        clip:
          - operation: AND
            vertices:
              - [-0.50, -0.50]
              - [-0.50, 0.50]
              - [0.50, 0.50]
              - [0.50, -0.50]

# Definition of the "glass_box" geometry that will be used
# by the glass
- geometry: &glass_box
    - material: *thin_glass
#     - material: *glass
      cuboid:
        size: [10,10,0.5]
      transform: { translation: [0, 0, 0.25] }

# Definition of the receiver's entity.
- entity:
    name: "square_receiver"
    primary: 0
# set "primary" to 0 because this entity should not the sampled as a primary reflector
    transform: { rotation: [0, 0, 0], translation: [0, 0, -10] }
# The "small_square" geometry is created as horizontal. In order to
# have it vertical, a rotation of 90 degrees around the Y-axis must be performed.
# Also, a translation is applied in order to position it in space.
    anchors:
      - name: "anchor0"
        position: [0, 0, 0]
# An anchor is defined, at a local position (in the referential of the square)
    geometry: *small_square

# Definition of a glass box
- entity:
    name: "glass_slide"
    primary: 0
    transform: { rotation: [0, 0, 0], translation: [0, 0, 0] }
    geometry: *glass_box

# Definition of the primary reflectors
- entity:
    name: "primary_reflector1"
    primary: 1 # primary=1 -> sampled for integration
    transform: { rotation: [0, 0, 0], translation: [0, 0, -2.0] }
    geometry:
    - material: *specular
      parabol:
        focal: 12
        clip:
        - operation: AND
          circle:
            radius: 10
            center: [0, 0]
        - operation: SUB
          circle:
            radius: 5
            center: [0, 0]

- entity:
    name: "primary_reflector2"
    primary: 1 # primary=1 -> sampled for integration
    transform: { rotation: [0, 0, 0], translation: [0, 0, -4] }
    geometry:
    - material: *specular
      parabol:
        focal: 14
        clip:
        - operation: AND
          circle:
            radius: 15
            center: [0, 0]
        - operation: SUB
          circle:
            radius: 10
            center: [0, 0]


- entity:
    name: "primary_reflector3"
    primary: 1 # primary=1 -> sampled for integration
    transform: { rotation: [0, 0, 0], translation: [0, 0, -6] }
    geometry:
    - material: *specular
      parabol:
        focal: 16
        clip:
        - operation: AND
          circle:
            radius: 20
            center: [0, 0]
        - operation: SUB
          circle:
            radius: 15
            center: [0, 0]


```

```
# Definition of the secondary reflector
- entity:
    name: "secondary_reflector"
    primary: 0  # primary=0 -> NOT sampled for integration
    transform: { rotation: [0, 0, 0], translation: [0, 0, 6] }
    geometry:
    - material: *specular
      hyperbol:
        focals: &hyperbol_focals { real: 16.0, image: 4 }
        clip:
        - operation: AND
          circle:
            radius: 5
            center: [0, 0]

# Atmospheric absorption properties
- atmosphere:
    extinction: *air_kabs
```

The following "horizontal_receiver.yaml" file is used to define the receiver:

```
- { name: "square_receiver", side: FRONT }
```

The "horizontal_receiver.yaml" file illustrates the use of the following new concepts:

- Spectra: at the beginning of the file, 4 different spectra are defined in order to describe the spectral dependence of: the incoming sunlight ("solar_spectrum"), the absorption coefficient of the atmosphere ("air_kabs"), the absorption coefficient of the glass ("glass_kabs"), and the refraction index of the glass ("glass_ref_index"). These spectra have been defined from 0.3 to 0.8 micrometers with a step of 0.1 micrometer, but any spectral discretisation can be used, with no limit on the number of wavelengths.

- Two media have been defined: air ("air_medium") and glass ("glass_medium"). Each medium needs the definition of its refraction index and its absorption coefficient. These properties can be a constant, such as in the case of the refraction index of air, or a spectrum.

- Two new categories of material properties are introduced: "thin_dielectric" and "dielectric" that are used respectively in the "thin_glass" and "glass" materials. In each case, the "medium_i" and "medium_t" properties need to be set using the previously defined "air_medium" and "glass_medium" media: "medium_i" refers to the medium on the exterior of the material, while "medium_t" refers to the medium on the interior of the material. Since "thin_dielectric" is a surface property, we do not need to distinguish between the front and back side properties, but we need to define the thickness of the material. However, "dielectric" is a volume property. But since we will use it over a geometry that is defined by its surface (see the "glass_box" geometry), we need to provide some information about what "medium_i" and "medium_t" refers to, for each side of this surface.

- The "cuboid" shape is used in order to define the "glass_box" geometric template. It is later used to define the "glass_slide" geometric entity.

- The "circle" shape is used in order to perform clipping operations over the three paraboloids and the hyperboloid.

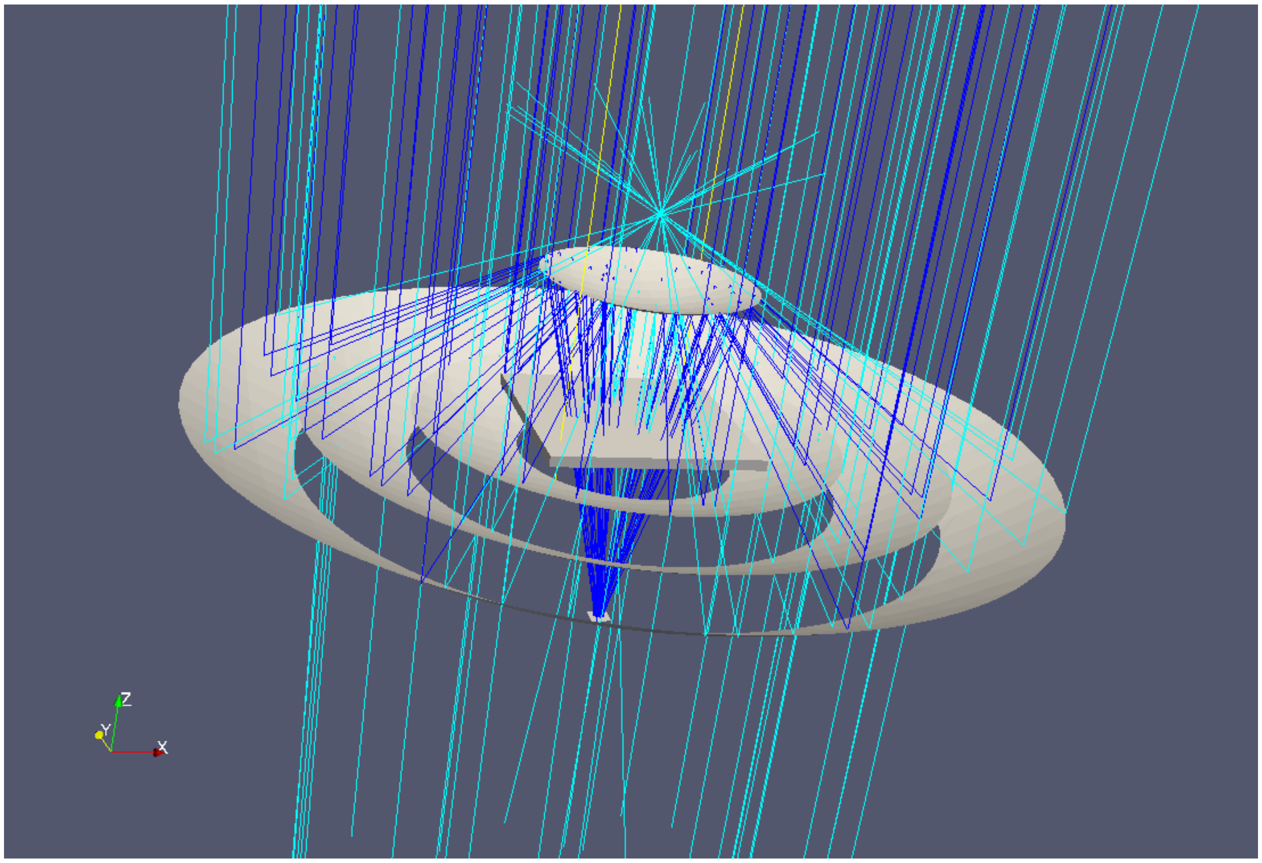

Figure 7: 3D visualisation of 100 ray paths in the example of a complex optical path space, in the case the box uses the "thin_dielectric" material.

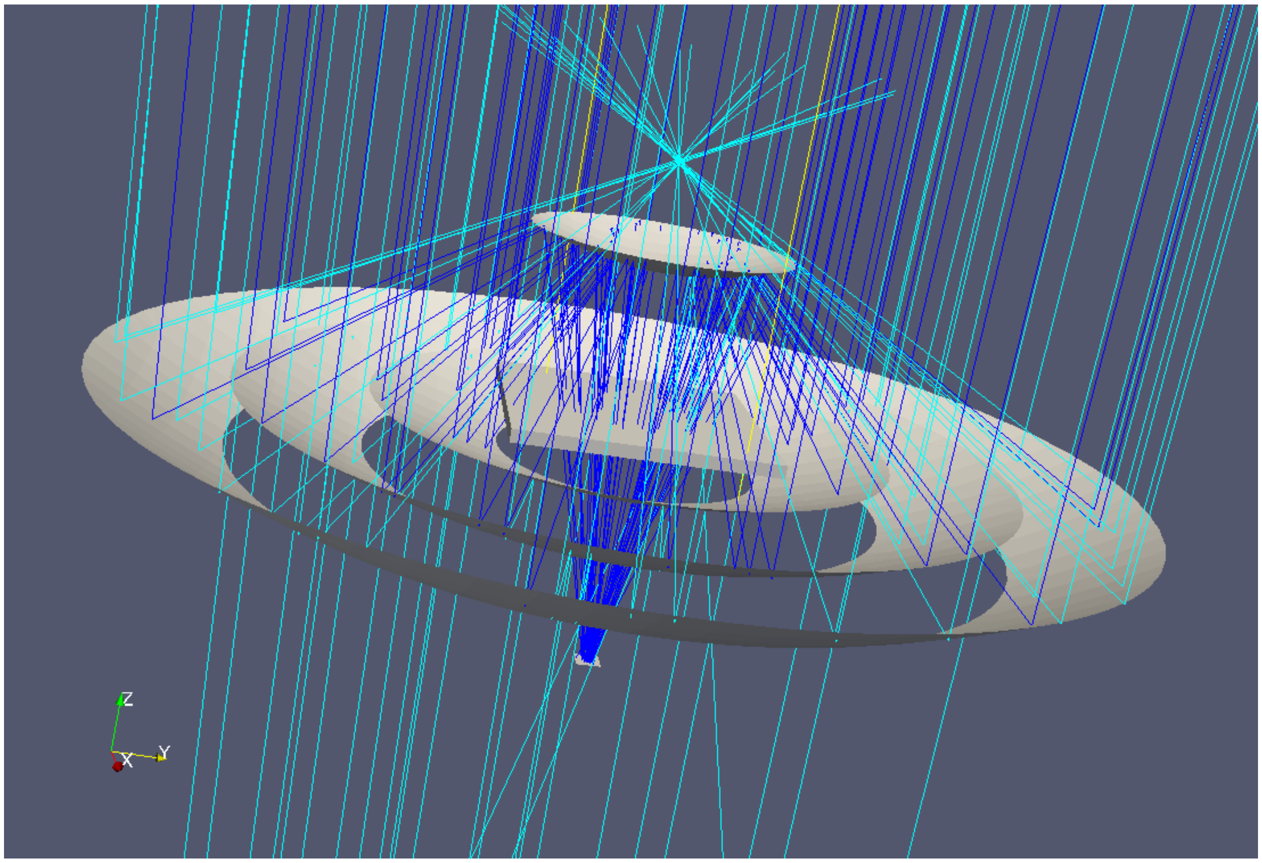

Figure 8: Identical to figure 7, but in the case the box uses the "dielectric" material.

- Finally, the absorption optical properties of the atmosphere are set, using the "air_kabs" spectrum that was defined at the beginning of the file.

Note that the "glass_box" template was defined using the "thin_glass" material (line 97). When running the **solstice** computation:

```
solstice -v -D 0,90 -R horizontal_receiver.yaml multiple_reflectors.yaml
```

The output is the following:

```
#--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
7 1 3 10000 0
1304.77 0
566.86 5.8624
0.901466 0
22.1126 1.5975
585.567 5.87917
0.011813 0.000179591
1.64295 0.0203065
square_receiver 10 1   566.86 5.8624   566.87 5.8625   568.312 5.8774   0.0103591
    0.000162878   1.45212 0.0204591   566.86 5.8624   566.87 5.8625   568.312 5.8774
    0.0103591 0.000162878   1.45212 0.0204591   0.434452 0.00449305   -1 -1   -1 -1
    -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
primary_reflector2 2 430.566 3329   0.901466 0   0.23551 0.166514
primary_reflector3 18 626.685 4642   0.901466 0   0.117683 0.117677
primary_reflector1 22 247.518 2029   0.901466 2.16927e-09   21.7595 1.58492
10 2   250.228 4.80559   250.233 4.80567   250.875 4.81801   0.00428719 0.000109752
    0.646888 0.0155304   250.228 4.80559   250.233 4.80567   250.875 4.81801
    0.00428719 0.000109752   0.646888 0.0155304   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
      -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
10 18   120.194 3.55664   120.196 3.55671   120.547 3.56706   0.00207403 7.8854e-05
    0.352295 0.0126445   120.194 3.55664   120.196 3.55671   120.547 3.56706
    0.00207403 7.8854e-05   0.352295 0.0126445   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
      -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
10 22   196.437 4.38091   196.441 4.381   196.89 4.39101   0.00399788 0.000123005
    0.452937 0.0123931   196.437 4.38091   196.441 4.381   196.89 4.39101   0.00399788
    0.000123005   0.452937 0.0123931   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
    -1 -1   -1 -1   -1 -1   -1 -1
```

We can see that from the 1304.77 watts that could potentially reach primary reflectors, only 1304.77*0.901466=1176.21 watts are really intercepted by primary reflectors, and 560.165±5.8 watts are finally absorbed by the receiver: 21.76±1.58 watts are lost because of shadowing, 592.65±5.9 watts are lost because their optical paths do not end on the receiver, 1.17e-2±1.80e-4 watts are absorbed during surface reflections and by semi-transparent solids (the glass), and 1.63±2.0e-2 watts are lost because of atmospheric absorption. Figure 7 shows a representation of 100 optical paths in the geometry.

We can try to change the material of the glass box: let us comment line 97 and uncomment line 98, thus defining the "glass_box" made of the "glass" material (that uses the "dielectric" material property). The output of the computation is the following:

```
#--- Sun direction: 0 90 (-6.12323e-17 -0 -1)
7 1 3 10000 0
1304.77 0
570.507 5.83915
```

```
0.901466 0
22.2294 1.6016
578.202 5.86881
3.68103 0.14479
1.57284 0.0195712
square_receiver 10 1   570.507 5.83915   573.014 5.86468   571.926 5.85361   2.50642
    0.0385616   1.41933 0.0198379   570.507 5.83915   573.014 5.86468   571.926 5.85361
        2.50642 0.0385616   1.41933 0.0198379   0.437247 0.00447524   -1 -1   -1 -1   -1
    -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1
primary_reflector2 2 430.566 3324   0.901466 0   0.23551 0.166514
primary_reflector3 18 626.685 4661   0.901466 0   0.117683 0.117677
primary_reflector1 22 247.518 2015   0.901466 2.12557e-09   21.8762 1.58905
10 2   259.355 4.8553   260.456 4.87583   260.007 4.86745   1.10059 0.0275051
    0.651655 0.0152761   259.355 4.8553   260.456 4.87583   260.007 4.86745   1.10059
    0.0275051   0.651655 0.0152761   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1
    -1   -1 -1   -1 -1   -1 -1
10 18   120.882 3.55773   121.386 3.57253   121.221 3.56768   0.504345 0.0194779
    0.33895 0.0122011   120.882 3.55773   121.386 3.57253   121.221 3.56768   0.504345
    0.0194779   0.33895 0.0122011   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1
    -1   -1 -1   -1 -1   -1 -1
10 22   190.27 4.31326   191.171 4.33363   190.699 4.32295   0.901478 0.0274126
    0.428725 0.0118504   190.27 4.31326   191.171 4.33363   190.699 4.32295   0.901478
    0.0274126   0.428725 0.0118504   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1 -1   -1
    -1   -1 -1   -1 -1   -1 -1
```

The modification in the nature of the glass material has the following consequences: it slightly increases the losses due to shadowing of primary reflectors, decreases the power that is not focused on the receiver; atmospheric absorption is not significantly modified, but losses due to absorption by materials (in this case, the glass) has been increased by factor 300 !

In these last two results sets, energy budgets can be verified: the sum of losses, plus the absorbed flux, is equal to the potential flux multiplied by the global cos-factor; the sum of per-primary shadow losses is equal to the global shadow losses; the sum of per receiver and per primary materials losses is equal to the global materials losses; the same is true for atmospheric absorption losses. Finally, the sum of per-primary cos-factors, pondered by the surface of each primary reflector, is equal to the global cos-factor.

5 Post-processing tools

Post-processing tools can be downloaded from the Méso-Star website (within the Solstice/Additional Resources section). Follow the instructions in order to compile these tools. Four different programs can be used:

5.1 solppraw

This program reformats the **Solstice** output into a much more human-friendly format. In can both be used over a ascii **Solstice** output file, or directly piped over the **Solstice** command. For instance, let it work over the console output of the last command used in the previous section:

```
solstice -v -D 0,90 -R horizontal_receiver.yaml multiple_reflectors.yaml | solppraw
```

This command should produce the ”0-90-raw-results.txt” file provided below:

```
 Overall results (#Samples = 10000)
-------------------------------------------------------------------------------------------
   Potential flux |         1304.77 +/- 0
    Absorbed flux |          566.86 +/- 5.8624
    Cosine factor |        0.901466 +/- 0
      Shadow loss |         22.1126 +/- 1.5975
     Missing loss |         585.567 +/- 5.87917
   Materials loss |        0.011813 +/- 0.000179591
Atmospheric loss |         1.64295 +/- 0.0203065

 Receiver 'square_receiver' (Area = 1)
------------------------------------------------------------[Front]-------------------------
                  |              [Incoming]                          [Absorbed]
             Flux |          566.86 +/- 5.8624           |          566.86 +/- 5.8624
    Material loss |       0.0103591 +/- 0.000162878      |       0.0103591 +/- 0.000162878
Atmospheric loss |         1.45212 +/- 0.0204591        |         1.45212 +/- 0.0204591
No Material loss |          566.87 +/- 5.8625           |          566.87 +/- 5.8625
   No Atmos. loss |         568.312 +/- 5.8774           |         568.312 +/- 5.8774
                  |
       Efficiency |        0.434452 +/- 0.00449305

 Primary 'primary_reflector2' (Area = 430.566; #Samples = 3329)
-------------------------------------------------------------------------------------------
    Cosine factor |        0.901466 +/- 0
      Shadow loss |         0.23551 +/- 0.166514

 Primary 'primary_reflector3' (Area = 626.685; #Samples = 4642)
-------------------------------------------------------------------------------------------
    Cosine factor |        0.901466 +/- 0
      Shadow loss |        0.117683 +/- 0.117677

 Primary 'primary_reflector1' (Area = 247.518; #Samples = 2029)
-------------------------------------------------------------------------------------------
    Cosine factor |        0.901466 +/- 2.16927e-09
      Shadow loss |         21.7595 +/- 1.58492

 Receiver 'square_receiver' X Primary 'primary_reflector2'
------------------------------------------------------------[Front]-------------------------
                  |              [Incoming]                          [Absorbed]
             Flux |         250.228 +/- 4.80559          |         250.228 +/- 4.80559
    Material loss |      0.00428719 +/- 0.000109752      |      0.00428719 +/- 0.000109752
Atmospheric loss |        0.646888 +/- 0.0155304        |        0.646888 +/- 0.0155304
No Material loss |         250.233 +/- 4.80567          |         250.233 +/- 4.80567
   No Atmos. loss |         250.875 +/- 4.81801          |         250.875 +/- 4.81801

 Receiver 'square_receiver' X Primary 'primary_reflector3'
------------------------------------------------------------[Front]-------------------------
                  |              [Incoming]                          [Absorbed]
             Flux |         120.194 +/- 3.55664          |         120.194 +/- 3.55664
    Material loss |      0.00207403 +/- 7.8854e-05       |      0.00207403 +/- 7.8854e-05
Atmospheric loss |        0.352295 +/- 0.0126445        |        0.352295 +/- 0.0126445
No Material loss |         120.196 +/- 3.55671          |         120.196 +/- 3.55671
   No Atmos. loss |         120.547 +/- 3.56706          |         120.547 +/- 3.56706

 Receiver 'square_receiver' X Primary 'primary_reflector1'
------------------------------------------------------------[Front]-------------------------
                  |              [Incoming]                          [Absorbed]
             Flux |         196.437 +/- 4.38091          |         196.437 +/- 4.38091
    Material loss |      0.00399788 +/- 0.000123005      |      0.00399788 +/- 0.000123005
Atmospheric loss |        0.452937 +/- 0.0123931        |        0.452937 +/- 0.0123931
No Material loss |         196.441 +/- 4.381            |         196.441 +/- 4.381
   No Atmos. loss |          196.89 +/- 4.39101          |          196.89 +/- 4.39101
```

5.2 solmaps

The **solmaps** command is used in order to produce a vtk file containing the required data for drawing a map or the incoming solar flux over all receivers that have been defined using the "per_primitive" flag.

For instance, let us use the last example of the last section. We first have to use the "per_primitive" flag over the "square_receiver" defined in the "horizontal_receiver.yaml" file:

```
- name: "square_receiver"
  side: FRONT
  per_primitive: INCOMING
```

We also need to modify the definition of the "square_receiver": if we want to draw a map of the incoming solar flux density, the receiver must be defined using more than the default 2 triangles (since the receiver is a square). In order to do that, we can use the "slices" parameter in the definition of the "small_square" geometry that is later used in the definition of the "square_receiver" entity; let us use for instance a value of 64 instead of the default (value of 1):

```
- geometry: &small_square
    - material: *black
      plane:
        slices: 64
        clip:
          - operation: AND
            vertices:
              - [-0.50, -0.50]
              - [-0.50, 0.50]
              - [0.50, 0.50]
              - [0.50, -0.50]
```

Then run **solstice** using a high number of samples, and pipe the command through the **solmaps** program:

```
solstice -n 1000000 -v -D 0,90 -R horizontal_receiver_variation1.yaml
    multiple_reflectors_variation2.yaml | solmaps
```

This produces the "0-90-square_receiver.vtk" file, that can finally be visualised using "paraview" in order to produce a map such as the one shown in figure 9

5.3 solpaths

The **solpaths** program uses the **solstice** output when it is invoked with the "-p" option (used for generating typical optical paths), in order to produce, for each input solar direction, a vtk file that can be opened in order to visualise the typical optical paths.

5.4 solpp

The **solpp** program will generate three files for each required input solar direction:

- A first vtk file where several results are mapped over primary geometries (reflectors).
- A second vtk file where various results are mapped over receivers.

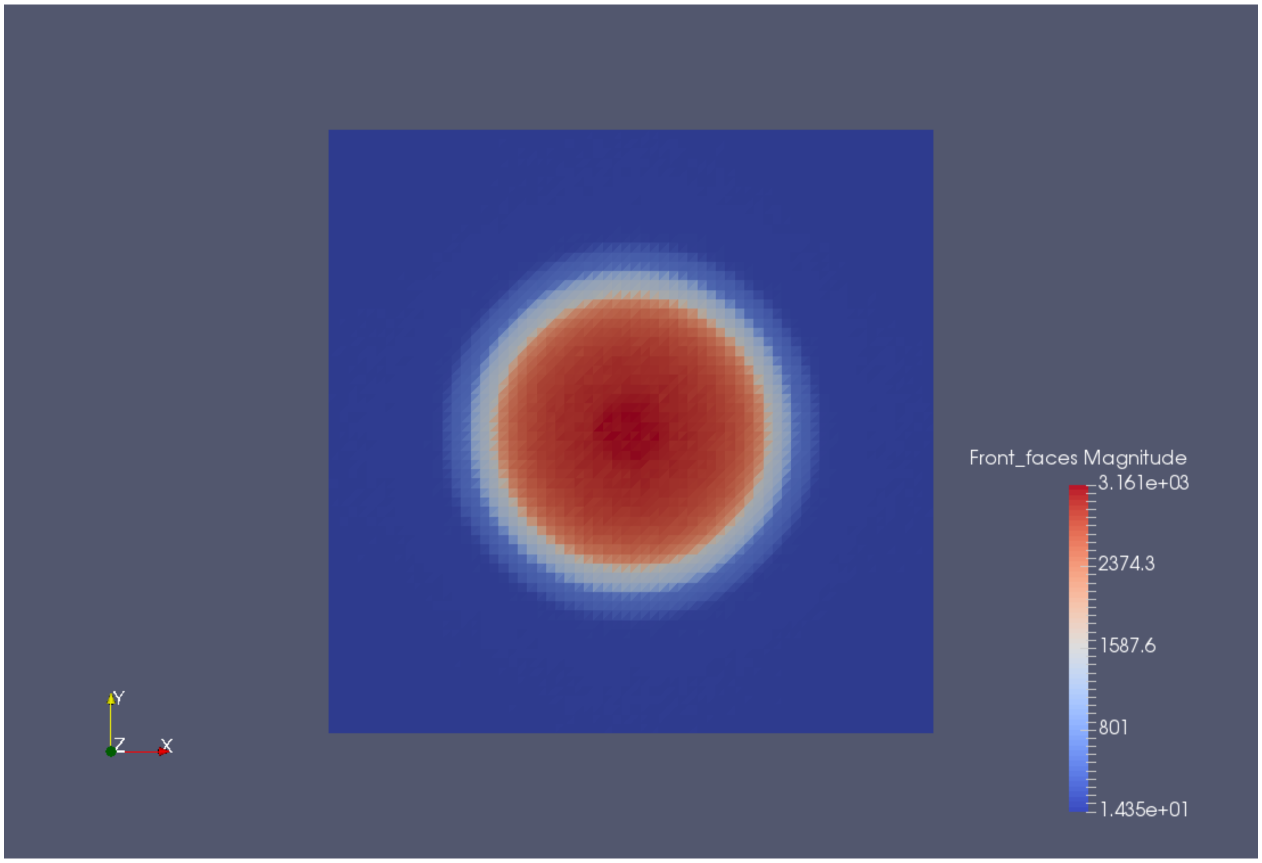

Figure 9: Map of the incoming solar flux density over the square receiver used in the last example of section 4

- A obj files used for the representation of geometries that are neither primary geometries, nor reflectors.

This program uses two input files:

- A legacy simulation ouput file. For instance, over the last example of section 4:

```
solstice -n 1000000 -D 0,90 -R horizontal_receiver.yaml -o simul.txt
    multiple_reflectors.yaml
```

- The geometry of a solar plant, produced by **solstice** when invoked with the "-g" options. For instance, over the last example of section 4:

```
solstice -D 0,90 -R horizontal_receiver.yaml -g format=obj-o geometry.obj
    multiple_reflectors.yaml
```

Then running the **solpp** command over the two files that result from previous commands:

```
solpp geometry.obj simul.txt
```

produces three files: "0-90-primaries.vtk", "0-90-receivers.vtk" and "0-90-miscellaneous.obj". Figure 10 shows the solar input flux over primary geometries, produces using the "0-90-primaries.vtk" file. Figure 9 is a good example of what can be produced from the "0-90-receivers.vtk" file. Figure 11 shows the geometry stored in the "0-90-miscellaneous.obj" file.

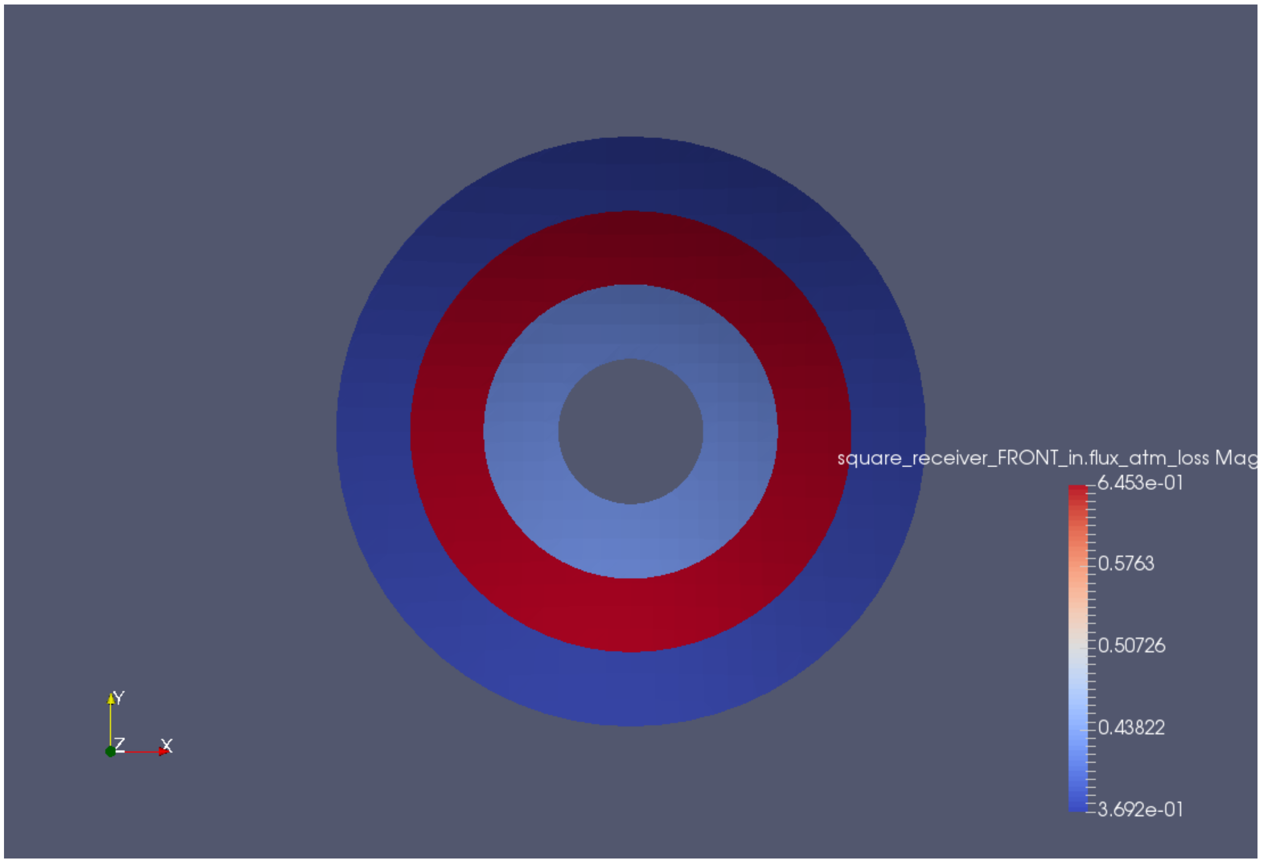

Figure 10: Map of the incoming solar flux density over the primary geometries used in the last example of section 4

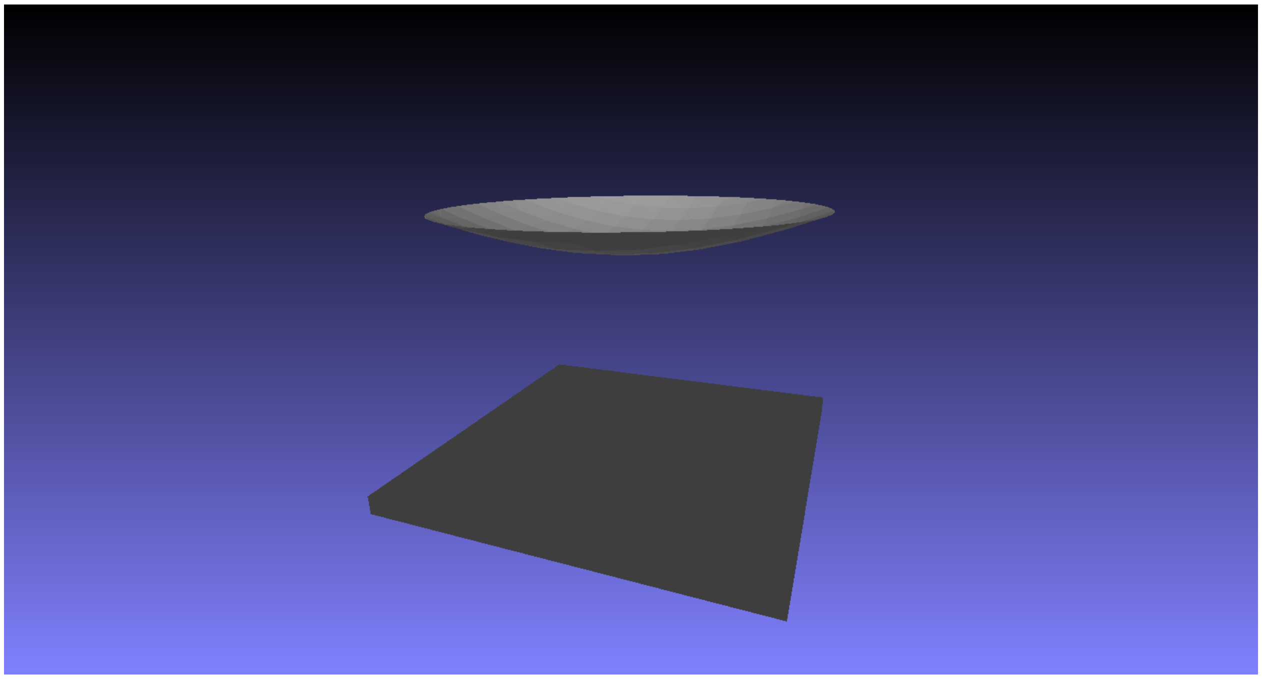

Figure 11: Geometries that are neither primary geometries, nor receivers, produced from the "0-90-miscellaneous.obj" file.

References

[1] D. Buie, A.G. Monger, and C.J. Dey. Sunshape distributions for terrestrial solar simulations. *Solar Energy*, 74:113–122, 2003.