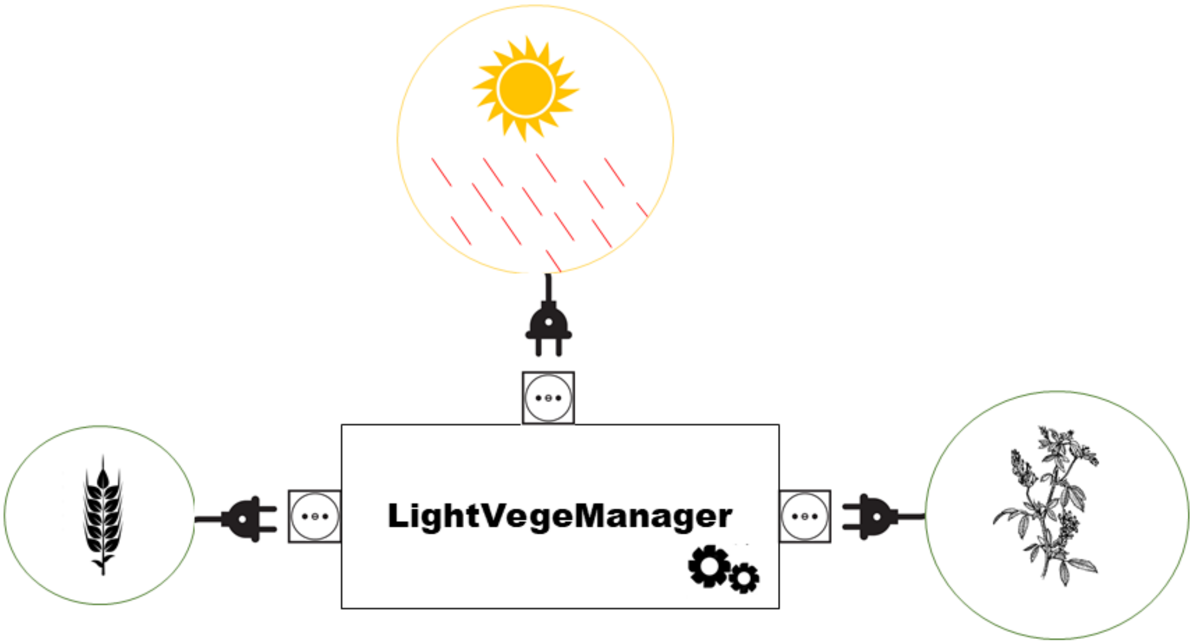
LightVegeManager

Release 0.0.0

INRAE P3F

Nov 27, 2023

USER GUIDE


LightVegeManager

CHAPTER

ONE

MODULE DESCRIPTION

Summary

Version

Date

Nov 27, 2023

Author

See *Authors* section

Overview

LightVegeManager is a Python package made for plant modeling and manage enlightment. It serves as an interface to merge several plant models and a light model. Applications involved by this tools are part of OpenAlea platform.

CHAPTER

TWO

DOCUMENTATION

2.1 Presentation

2.1.1 Context

This package was born in the framework of the research project MobiDiv (https://anr.fr/ProjetIA-20-PCPA-0006), which aims to find solutions for pesticide-free agriculture. One of the targets of this project is to model coupled crops between wheats, modelled by CN-Wheat (https://www.quantitative-plant.org/model/CN-Wheat), and alfalfas, thanks to l-egume model (https://github.com/glouarn/l-egume). Thus, LightVegeManager was created to manage the lighting in a vegetal simulation, in formats understandable by the tools proposed on the OpenAlea platform (https://github.com/openalea).

2.1.2 Intentions

We want to simplify the light management when several models, plant and lighting, are involved. The tool manages many input and output formats, in order to match a wide variety of situations.

2.1.3 How

It reads the following input formats:

- PlantGL scene
- MTG table
- VGX file
- Caribu scene, dict storing a triangulation
- Dict storing a grid of voxels with leaf area and leaf angle distribution

More details about the input formats here *Scenes format*.

And it can return the outputs in the following formats:

- Pandas Dataframe, in multiple scales (triangles, voxels, elements) depending on the light model.
- updates a MTG table
- returns two tables compatible with l-egume

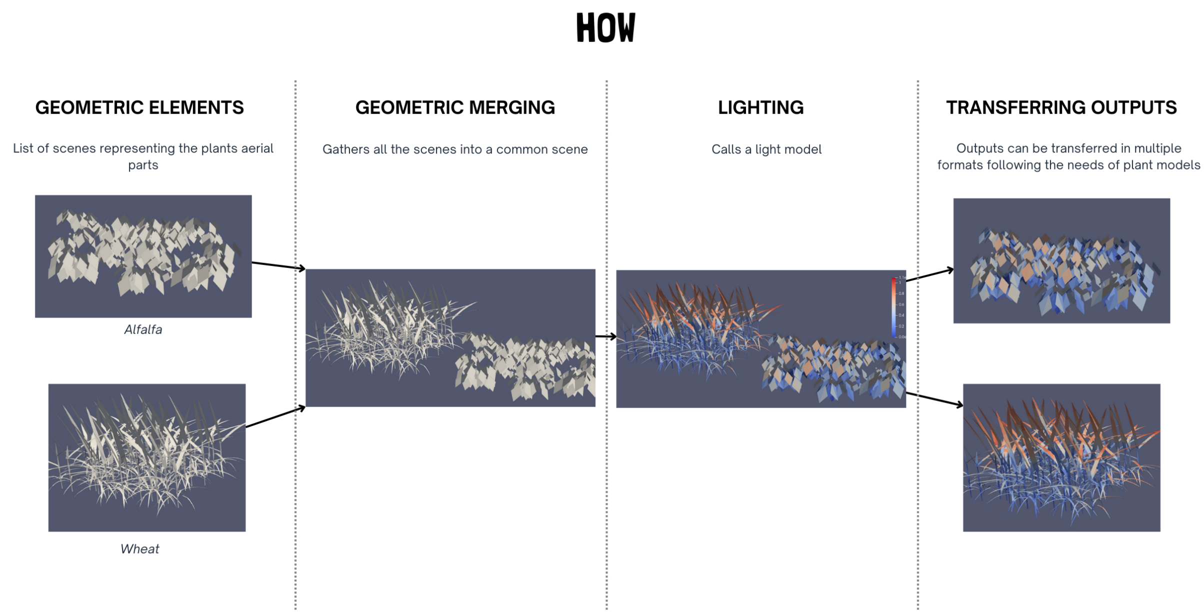


Fig. 1: General Functionning

2.1.4 Lighting

At the moment, LightVegeManager can handle three lighting models, CARIBU (https://github.com/openalea-incubator/caribu), PyRATP (https://github.com/openalea-incubator/PyRATP), and RiRi modified in l-egume (https://github.com/glouarn/riri5). Those models are ajusted for virtual plant canopies with considerations, like infinitisation of the scene, which are preferred for simulating large crop fields.

They offers two different approach for light modelling:

Surfacic approach (CARIBU)

The radiations are computed on a set of triangles. This approach offers a high precision of lighting on the different organs, but has a high cost in time computation.

Volumic approach (PyRATP, RiRi)

The geometric scene is represented by a set of voxels, which contains a leaf area and a leaf angle informations. Then, the models solves turbid medium equations from the rays inside the grid. This approach has a higher approximation of geometry, but is well adapted for dense canopy. The computation cost is highly reduced regarding the surfacic approach but the geometric scene must matches the turbid medium representation and hypothesis.

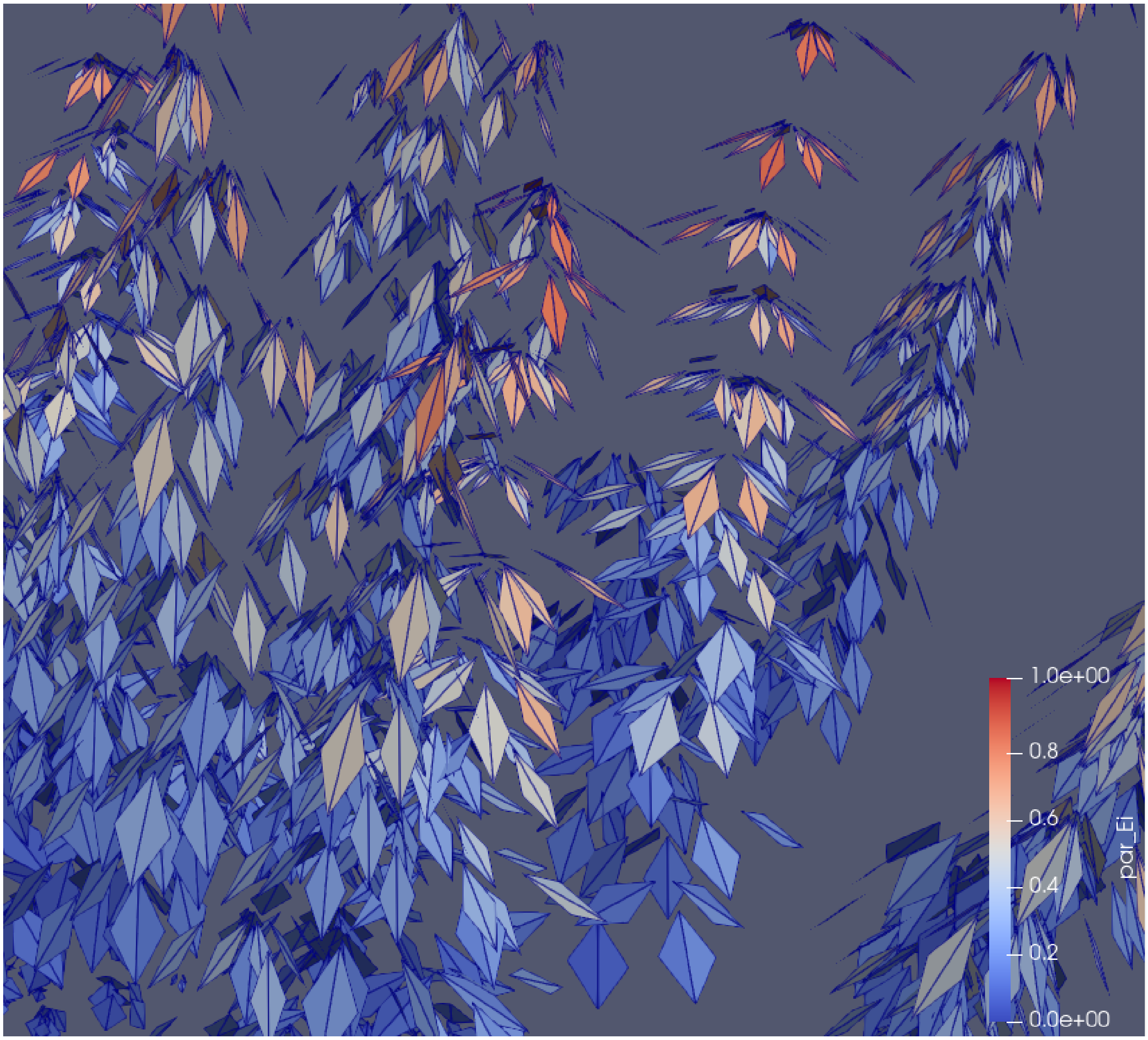


Fig. 2: Example of a triangulation

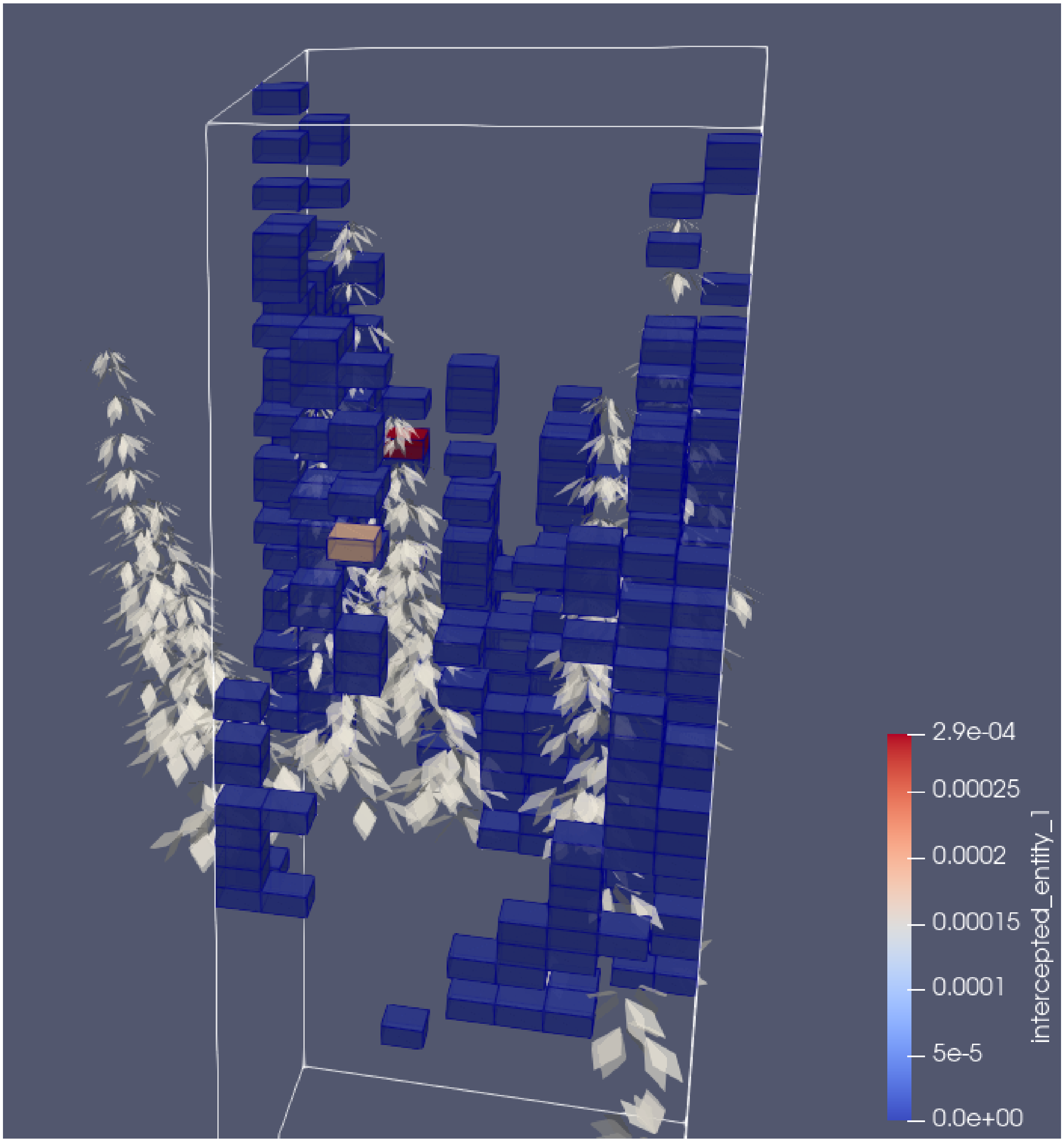


Fig. 3: Example of a grid of voxels

2.2 Installation

2.2.1 Setting a Python environment

We recommend an installation with conda.

1. Create a conda environment with miniconda3

```
conda create -n myenvname python=3 openalea.mtg openalea.plantgl alinea.
↪caribu alinea.astk numpy=1.20.3 pandas pytest sphinx sphinx-rtd-theme -c␣
↪conda-forge -c openalea3
```

2. Place yourself in the created environment: `conda activate myenvname`

Note: `virtual sensors` and `soilmesh` in Caribu > 8.0.10 are working correctly

2.2.2 Installation of Needed Packages

PyRATP

1. Git console:

```
git clone -b update_mobidiv https://github.com/mwoussen/PyRATP
```

2. Installation in the conda environment (in folder `PyRATP`)

```
make mode=develop
make clean
```

RiRi5

1. Git console:

```
git clone https://github.com/glouarn/riri5
```

2. Installation in the conda environment (in folder `riri5`)

```
python setup.py develop
```

LightVegeManager

1. Git console:

```
git clone https://github.com/mwoussen/lightvegemanager
```

2. Installation in the conda environment (in folder `lightvegemanager`)

```
python setup.py develop
```

2.2.3 Quickstart

Small test to quickly test the tool. LightVegeManager needs at least a geometric information and a light model to call between "ratp", "caribu" or "riri5".

```
from lightvegeamanger.LVM import LightVegeManager

# one triangle as a geometric element
# we write our triangle in a CaribuScene format
organ_id = 001
triangle_vertices = [(0,0,0), (1,0,0), (1,1,1)]
triangle = {organ_id : [triangle_vertices]}
geometry = { "scenes" : [triangle] }

# surfacic lighting with CARIBU
lighting = LightVegeManager(lightmodel="caribu")

# build the scene
lighting.build(geometry)

# compute lighting
energy = 500
hour = 15
day = 264 # 21st september
lighting.run(energy, hour, day)

# output
print(lighting.elements_outputs)
```

See also:

For more details on default values, see `LightVegeManager_defaultvalues`

2.3 Architecture of the tool

2.3.1 Package Content

The package has 8 folders and 4 files in its root:

- data: data files used in some the use examples in the package
- doc: user and reference documentations
- notebooks: jupyter notebooks with documented tutorials exploring the tool features
- s2v and s5: sources of analysis tools
- src: sources of LightVegeManager
- tests: unit testing files written in pytest format
- .gitignore: files and folders to ignore by git
- .readthedocs.yaml: config file for read-the-docs
- requirements.txt: packages needed for read-the-docs in order to compile
- README.md: Read Me file

- pytest.ini: config file for pytest
- setup.py: setup script used for installation

2.3.2 Code structure

The code structure of the tool is designed with a main class LightVegeManager, which calls modules for computing specific parts depending on the inputs.

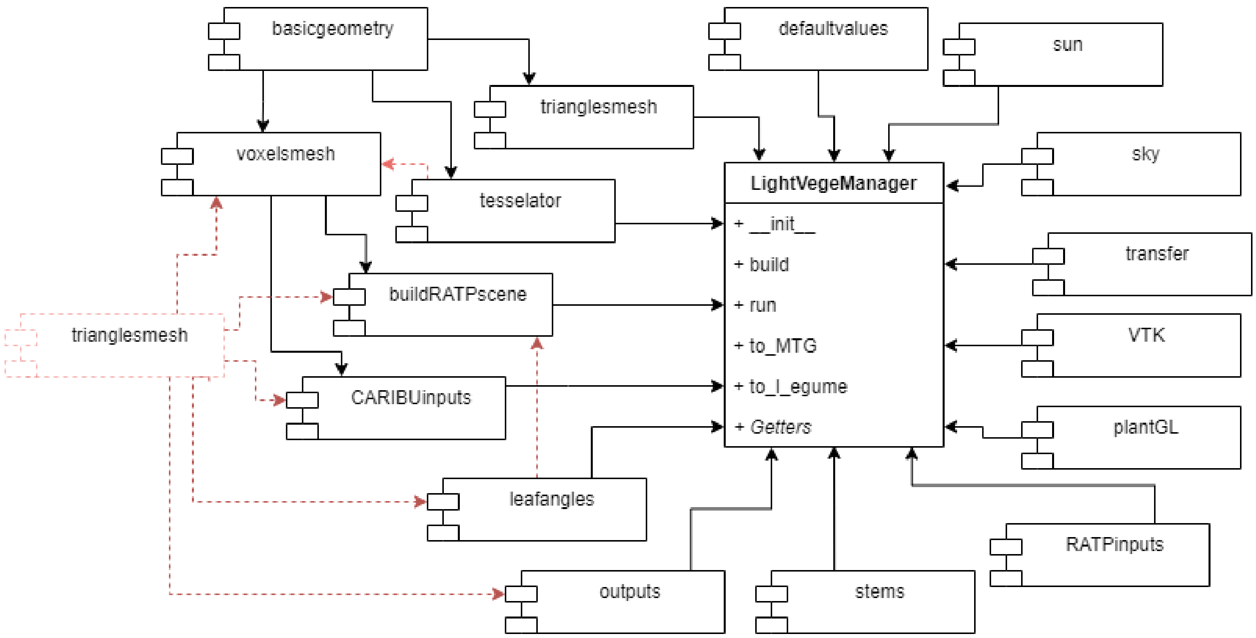


Fig. 4: Dependances between modules

Modules are:

- `LightVegeManager_tool` Main class of the tool. Calls all the other modules in `src`.
- `LightVegeManager_CARIBUinputs` Manages and prepares input information for CARIBU.
- `LightVegeManager_RATPinputs` Manage vegetation and meteo input informations for RATP
- `LightVegeManager_RiRi5inputs` Manage grid of voxels for RiRi5
- `LightVegeManager_VTK` Writes VTK files from LightVegeManager geometry and lighting data. Used for visualisation
- `LightVegeManager_basicgeometry` Provides basic geometric operations in 3D used by LightVegeManager.
- `LightVegeManager_buildRATPscene` Build a PyRATP.grid from inputs.
- `LightVegeManager_defaultvalues` Default for simulation fixed parameters
- `LightVegeManager_leafangles` Handles leaf angle distribution, both in its dynamic computing or reading a file
- `LightVegeManager_outputs` Manages and reformats output results from the light models in pandas.Dataframe with similar columns names
- `LightVegeManager_plantGL` Visualisation with plantGL
- `LightVegeManager_sky` Build a sky from LightVegeManager input informations

- `LightVegeManager_stems` Manages stems element
- `LightVegeManager_sun` Build a sun respecting each light model format
- `LightVegeManager_tesselator` Manages subdivision of a triangulation
- `LightVegeManager_transfer` Manages transfer of LightVegeManager results to plant Models
- `LightVegeManager_trianglesmesh` Builds and handles triangulation mesh.
- `LightVegeManager_voxelsmesh` Builds and handles axis oriented voxels mesh

2.3.3 Front End: the main commands

th tool is used through the class `LightVegeManager` and the following methods:

- constructor `__init__` builds the sky, which stays the same throughout all the simulation. It sets also default values if not precised in the inputs.
- `build()` creates a common geometric scene from inputs and set parameters for the light model.
- `run()` computes the lighting.

The outputs from radiations are automatically gathered in a pandas Dataframe and accessible from the getters `elements_outputs()`, `triangles_outputs()` and `voxels_outputs()`.

As part of our initial objective, we added two methods in order to convert the results in formats understandable by CN-Wheat and l-egume:

- `to_MTG()`, which updates a MTG table read by CN-Wheat
- `to_l_egume()`, which updates two tables read by l-egume

Note: l-egume needs a local information of transmitted lighting among a voxel grid. Then, you need to provide the grid dimensions to LightVegeManager.

The other getters available are:

- `sensors_outputs()`, outputs of virtual sensors, only with CARIBU
- `soilenergy()`, radiation received by the soil, only with CARIBU
- *`sun()`*, object containing sun position xyz
- `maxtrianglearea()`, if you entered a triangle mesh, return the largest triangle
- `tesselationtime()`, if you activated tesselation of a triangle mesh (redraw of triangles), return computation time
- `modelruntime()`, return the computation time of the light model

Finally, you also have additional tools available for analysing the inputs and visualising the outputs (*additional tools*).

2.3.4 Back End: More details about how the tool works

First of all, the geometric merging is set in a Caribu scene format. It is a dict where keys are indices and values are list of triangles, one triangle is list of 3 3-tuple representing the vertices. Here is an example:

```
# organ 1
triangle1 = [(0,0,0), (0,1,0), (1,1,0)]
triangle2 = [(0,0,0), (0,1,1), (1,1,1)]

# organ 2
triangle3 = [(0,0,2), (0,1,2), (1,1,2)]

caribuscene = { 1 : [triangle1, triangle2] ,
                2 : [triangle3] }
```

We choose this format for its low processing cost, because it uses basic python objects.

We also save and recreate a dict to organize the indices inside each input scene called `matching_ids` (*index managment*).

The input parameters defines a common way for setting each light model parameters.

Geometric merging

We built a module to tesselate one triangle into four smaller triangles. The tesselation is applied either uniformly among all the triangle mesh or on a sides of a voxels grid. The tesselation following a grid is made in order to have a better matching of triangles in a voxels grid and attenuate the error of converting the mesh type.

We also built the possibility to compute dynamicaly the leaf angle distribution, either globally among all triangles, or locally inside each voxel.

Managing indexes

LightVegeManager expects a list of geometric scenes in its inputs. Each scene represents a plant specy. Each scene can also by sorted by elements, which can represent plant organs or just sets of triangles. The tool will then reorder all the indexes in order to avoid any confusion. The correspondance between input indices and new indices is stored in the dict attribute `matching_ids`.

Example of reordering:

Triangle subdivision

We implemented an algorithm for subdividing a triangle in 4 triangles according to triangle position in a grid of voxels. The goal was to have a better matching between a triangulation and a grid of voxels. Triangles are subdivided if they are between several voxels.

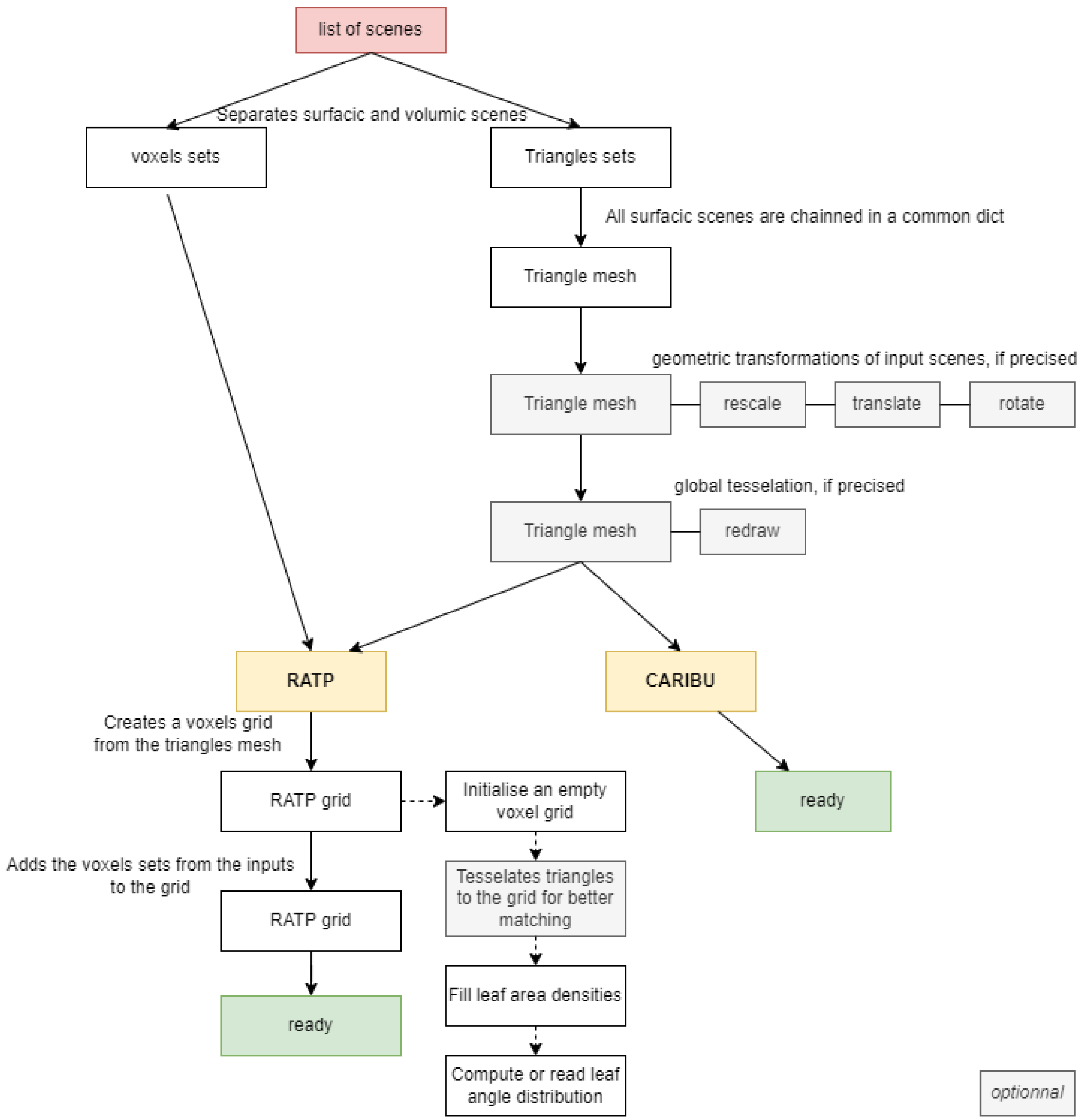


Fig. 5: Tool workflow for geometry

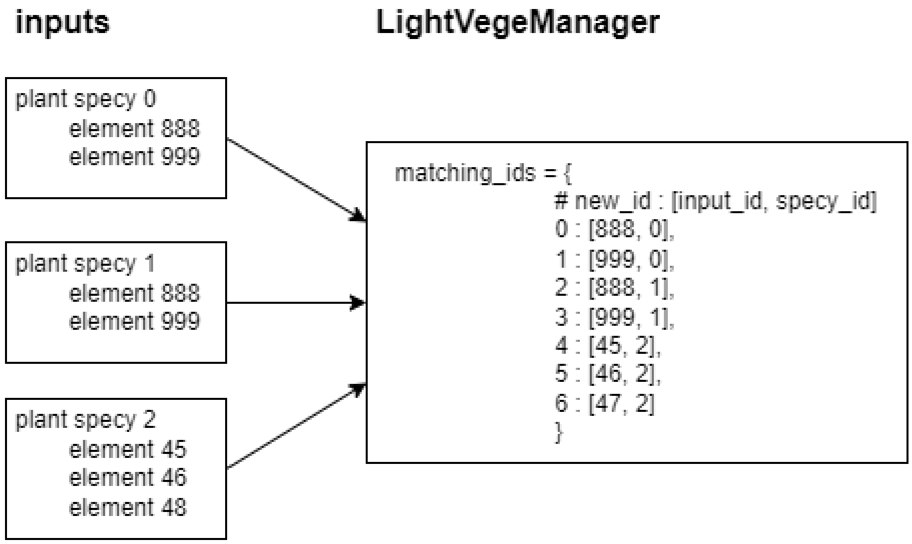


Fig. 6: Reordering indices in LightVegeManager

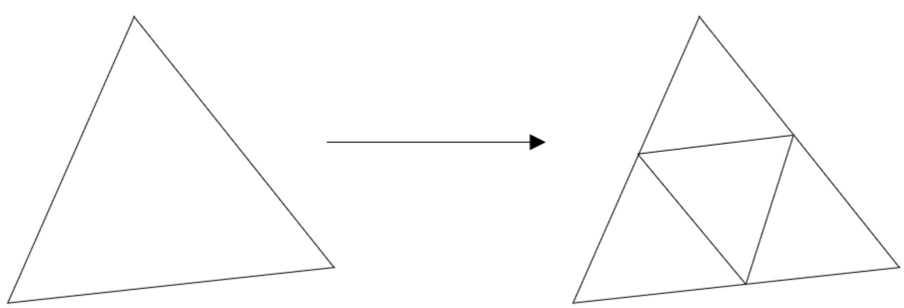


Fig. 7: Subdivision of a triangle

CARIBU, l-egume and virtual sensors

l-egume needs two different values to understand lighting results:

- total intercepted radiation for each plant
- local transmitted radiation following a voxel grid

In order to use CARIBU with l-egume, you need to retrieve transmitted radiations for each position of a voxel grid. LightVegeManager implements functions which can create a set of virtual sensors following a voxel grid. Then, with the virtual sensors and the soilmesh options activated, you can calculate the local transmitted radiation. Make sure to have the same grid dimensions as l-egume intern grid.

2.4 Inputs

In LightVegeManager, we choose to organize the input parameters in 3 python dict: geometric, environment and light model parameters.

2.4.1 Geometric inputs

```
geometry = {
                        "scenes" : [scene0, scene1, scene2, ...] ,
                        "domain" : ((xmin, ymin), (xmax, ymax)),
                        "stems id" : [(id_element, id_scene), ...],
                        "transformations" : {
                                                                                                "scenes unit" :␣
↪kwarg ,
                                                                                                "rescale" : kwarg ,
                                                                                                "translate" : kwarg ,
                                                                                                "xyz orientation" :␣
↪kwarg
                                                                                                }
            }
```

It contains the geometric scenes and directives to operate on them. You can also add descriptions on the scenes, especially if you have a large numbers of scenes and plant model evolved. Each scene must contains only one specy of plant, except for l-egume grid format, which can contains multiple species. As the geometric informations are likely to change during a simulation, this dict is not read by the constructor but rather by the `build()` method.

Scenes

`"scenes"` key needs a list of the geometric scenes. Each scene represents one plant specie, except for voxel grid format. There is no restrictions on the numbers of input scenes. Possible format are:

- PlantGL scene
- MTG table
- VGX file
- CaribuScene dict

```
# a vertice is a 3-tuple of float
triangle1 = [ (x1, y1, z1), (x2, y2, z2), (x3, y3, z3) ]
triangle2 = [ (x4, y4, z4), (x5, y5, z5), (x6, y6, z6) ]
mycaribuscene = { 888 : [ triangle1, triangle2 ] }

# 888 is the id of the element composed by triangle1 and triangle2
```

- Voxels grid: a dict with two keys:
  - `"LA"`: a `scipy.array` of dimension `[number of species, nz, ny, nx]` and stores for each voxel of each specie a leaf area
  - `"distrib"`: a list of `scipy.array` for each specy storing a leaf angle distribution

`"domain"` is necessary for CARIBU and infinite scene, it defines the xy square to replicate. `"stems id"` indicates the organ id of possible stems. They will be considered as strictly opaque element.

Geometric transformations

There is an optional fourth input key in the geometric dict, used to specify geometric transformations on several scenes. For example, you can arrange your inputs scenes in a specific pattern by translating them. The value of each entry is a dict:

```
"transformation" : { index_from_input_scenes : transformation_value}
```

Possible keys are the following:

`"scenes unit"`

Associates a measure unit among this list `['mm','cm','dm','m','dam','hm','km']` with a scene. You can also assign a unit to the inside global scene, the tool will rescale all the scenes in the same unit.

example:

```
transformations["scenes unit"] = {0 : 'm',
                                  1 : 'cm',
                                  2 : 'hm'}
```

`"rescale"`

Multiply all the lengths of a scene by a ratio.

example:

```
transformations["rescale"] = {0 : 0.5,
                              2 : 5}
```

"translate"

Apply a translation vector to a scene. A vector is 3-tuple.

example:

```
transformations["translate"] = {0 : (5, 0.2, 1),
                                1 : (0, 5, 0)}
```

"xyz orientation"

Change the orientation of a scene among the xyz axis. When a model creates a scene in a different convention used by the light models, LightVegeManager allows for a common convention. RATP and CARIBU use the convention x+ matches to North.

Possible convention:

- `"x+ = S"`: x+ matches to South
- `"x+ = W"`: x+ matches to West
- `"x+ = E"`: x+ matches to East

example:

```
transformations["xyz orientation"] = {0 : "x+ = S",
                                      1 : "x+ = W"}
```

2.4.2 Environment parameters

```
environment = {
                "coordinates" : [latitude, longitude, timezone] ,

                "sky" : "turtle46" ,
                "sky" : ["file", filepath] ,
                "sky" : [nb_azimut, nb_zenith, "soc" or "uoc"] ,

                "direct" : bool, # sun radiations
                "diffus" : bool, # sky radiations
                "reflected" : bool, # reflected radiation in the canopy
                "infinite" : bool, # infinitisation of the scene
                }
```

Contains parameters related to the environment of modelling and static parameters during all the simulation.

The sky is defined by the `"sky"` key and has 3 different possibilities:

- `"turtle46"` : SOC sky with 46 directions in "turtle" shape
- `["file", filepath]` : sky defined by the file `filepath`
- `[nb_azimut, nb_zenith, "soc" ou "uoc"]`: computes dynamically the source directions

2.4.3 Light model parameters

Specific parameters for each light model, as they use very different syntax and approach.

CARIBU: surfacic approach

```
caribu_args = {
                    "sun algo" : "ratp",
                    "sun algo" : "caribu",

                    "caribu opt" : {
                                    band0 = (reflectance, transmittance),
                                    band1 = (reflectance, transmittance),
                                    ...
                                    },
                    "debug" : bool,
                    "soil mesh" : bool,
                    "sensors" : ["grid", dxyz, nxyz, orig, vtkpath, "vtk"]
                }
```

The key "sun algo" specifies how to calculate the sun position. Possibilities are "ratp" or "caribu", which will call the function related to either one of the light models.

"caribu opt" defines the optical parameters in CARIBU format. For example:

```
caribu_parameters["caribu opt"] = {
                                    "par" = (0.10, 0.07) ,
                                    "nir" = (0.15, 0.05)
                                    }
```

For virtual sensors, we propose the construction following a grid of voxels, where a sensor is the bottom side of a voxel.

```
caribu_parameters["sensors"] = ["grid", dxyz, nxyz, orig, path, "vtk"]
```

Where:

- `dxyz = [x, y, z]` : list of size of each side of a voxel
- `nxyz = [nx, ny, nz]` : list of number of voxels in each direction
- `path` : string of file path where to save the vtk file of the sensors
- `"vtk"` : flag to specify you want to write sensors in VTK format for visualisation

`path` and `"vtk"` are optional elements in the list

Note: At the moment, only grid option is available for sensors

RATP: volumic approach

```
ratp_args = {
                # Grid specifications
                "voxel size" : [dx, dy, dz],
                "voxel size" : "dynamic",

                "full grid" : bool,

                "origin" : [xorigin, yorigin, zorigin],
                "origin" : [xorigin, yorigin],

                "number voxels" : [nx, ny, nz],
                "grid slicing" : "ground = 0."
                "tesselation level" : int

                # Leaf angle distribution
                "angle distrib algo" : "compute global",
                "angle distrib algo" : "compute voxel",
                "angle distrib algo" : "file",

                "nb angle classes" : int,
                "angle distrib file" : filepath,

                # Vegetation type
                "soil reflectance" : [reflectance_band0, reflectance_band1, ...],
                "reflectance coefficients" : [reflectance_band0, reflectance_band1, ...],
                "mu" : [mu_scene0, mu_scene1, ...]
            }
```

Grid specifications

- `"voxel size"`: defines the size of one voxel, 2 possibilities
  - `[float, float, float]`: for a length on each direction (xyz)
  - `"dynamic"`: create squared voxels where the length of a side is l = 5.max(area(triangle))
- `"number voxels"`: `[nx, ny, nz]`, defines the number of voxels on each xyz axis. If not specified, the tool computes dynamically the grid size according to input geometries.
- `"origin"`: possibility to define the grid origin.
  - `[xorigin, yorigin, zorigin]`
  - `[xorigin, yorigin]` and `zorigin = -zmin`
- `"grid slicing"`: if `"grid slicing" : "ground = 0."` and `"number voxels"` is not specified and there are triangles in the input geometries, it rescales the grid to avoid the soil layer < 0
- `"tesselation level"`: if there are triangles in the input geometries, you can tesselate them a certain number of times to match the voxel grid.
- `"full grid"`: option to fill all empty voxels with a `1e-10`.

Leaf angle distribution

You can specify how the leaf angle distribution is computed. It can be computed dynamically from a triangulation.

- `"angle distrib algo"`: 3 possibilities
  - `"compute global"`: LightVegeManager computes a global leaf angle distribution
  - `"compute voxel"`: LightVegeManager computes a local leaf angle distribution in each voxel of the grid
  - `file`: LightVegeManager reads a global leaf angle distribution in a file
- `"nb angle classes"`: if `"angle distrib algo" = "compute global"` or `"angle distrib algo" = "compute voxel"`, number of angle classes in the distribution
- `"angle distrib file"`: if `"angle distrib algo" = file"` string precising the file path to read

Vegetation type

- `"soil reflectance"`: if precised, soil reflects rays by a coefficient, one float by wavelength `[coef_par, coef_nir]`
- `"reflectance coefficients"`: if precised, leaf reflects rays by a coefficient, one float by wavelength `[coef_par, coef_nir]`
- `"mu"`: clumping effect for each input species

2.5 Additionnal Functionalities

Here are more functionalities implemented in the tool.

2.5.1 Inputs analysis Tool

For computing a leaf area distribution, you can run s5 or s2v tools, which was developped in previous projects. This tools are accessible with the methods `s5()` and `s2v()`. Those tools will create output files with leaf angle distribution, leaf area density and other informations related to turbid medium, from a triangles mesh inside a LightVegeManager object.

2.5.2 Visualisation

VTK files

You can create VTK files from the geometric information you have in the inputs. Files can be either triangles or voxels mesh. Available methods

- `VTK_nolight()`: write only geometric informations from the common scene
- `VTK_light()`: write geometric informations associated with radiation values for each element
- `VTK_sun()`: write a line representing the sun position

PlantGL visualizer

You can export a plantGL scene from geometric scene inside an isntance. Available methods

- `plantGL_nolight()`: write only geometric informations from the common scene
- `plantGL_light()`: write geometric informations associated with radiation values for each element
- `plantGL_sensors()`: write virtual sensors stock in the instance

2.6 Tutorials with jupyter notebooks

Jupyter notebooks are available with documented tutorials exploring the tool features.

2.6.1 LightVegeManager first steps

Content:

- One triangle
  - CARIBU
  - RATP
- Set of triangles
  - CARIBU
  - RATP

Introduction

This notebook provides an introduction to the tool and its default parameters. Visualization is provided with PlantGL and and its adapation to jupyter notebook plantgl-jupyter.

The main methods of a LightVegeManager instance are : - constructor `__init__`: returns an instance and initialize static parameters such as sky and default parameters - `build`: build and arrange the geometric scene following the chosen light model format - `run`: compute lighting - DataFrame outputs are stored in `triangles_ouputs`, `voxels_ouputs` and `elements_ouputs`

[1]:
```python
# imports the LightVegeManager object and SceneWidget for visual representation
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
```

One triangle

As a first example, we can compute lighting on a single 3D triangle. A triangle is reprented with a list of 3 cartesian points (x, y, z)

[2]:
```python
t = [(0., 0., 0.), (1., 0., 0.), (1., 1., 1.)]
```

CARIBU (surfarcic modelling)

1) Initialize the instance. The `lightmodel` must be entered, at the moment you can choose between `"caribu"`, `"ratp"` and `"riri5"`

[3]:
```python
# initialize the instance
lighting = LightVegeManager(lightmodel="caribu")
```

2) Build the geometry. The triangle will be save inside the instance.

[4]:
```python
# build the scene
lighting.build(geometry=t)
```

In order to visualize the scene in the instance, you can give a plantGL Scene in SceneWidget through two methods: - `plantGL_nolight`: plots only geometric elements - `plantGL_light`: plots geometric elements and colors the scene according to PAR values

[5]:
```python
SceneWidget(lighting.plantGL_nolight(), size_display=(600, 400), plane=True, size_world␣
↪= 4, axes_helper=True)
```

[5]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'F8IKbCRRTrC164vqQvjmte6n0', 'data': b'x\
↪xdaSLrw\xf5\xf7e`Pp\xe0\...
```

3) Compute the lighting. By default it will compute direct and diffuse lighting.

[6]:
```python
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

Then, you can print the Dataframe outputs

[7]:
```python
# print the outputs
print(lighting.triangles_outputs)
```

```
   Day  Hour  Triangle  Organ  VegetationType      Area    par Eabs  \
0  264    15         0      0               0  0.707107  397.268686

       par Ei
0  467.374925
```

RATP

To use RATP, you need to create a new instance. The others methods can be used in the same way as with CARIBU

[9]:
```python
# initialize the instance
lighting = LightVegeManager(lightmodel="ratp")

# build the scene
lighting.build(geometry=t)
```

With `plantGL_nolight` you can precise if you want to plot the voxels. By default, if not precised, a voxel side is set as 3 times the longest triangle side.

[11]:
```python
# visualisation
SceneWidget(lighting.plantGL_nolight(printvoxels=True),
            size_display=(600, 400),
            plane=True,
            size_world = 4,
            axes_helper=True)
```

[11]:
```
SceneWidget(axes_helper=True, scenes=[{'id': '7Btw3DEYYw3MHgwE6O2FD7CTm', 'data': b'x\
↪xdaSLrw\xf5\xf7e`Pp\xe0\...
```

[12]:
```python
# compute the lighting
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

You can get the outputs of the voxels or the triangles

[13]:
```python
# print the outputs
print(lighting.voxels_outputs)
```

```
   VegetationType    Day  Hour  Voxel  Nx  Ny  Nz   ShadedPAR   SunlitPAR  \
0             1.0  264.0  15.0    1.0   1   1   1  380.635895  473.640411

   ShadedArea  SunlitArea      Area        PARa  Intercepted  Transmitted
0    0.010761    0.696346  0.707107  472.225067     0.667827     8.742233
```

[14]:
```python
# print the outputs
print(lighting.triangles_outputs)
```

```
   Triangle  Organ  Voxel  VegetationType  primitive_area    Day  Hour  Nx  \
0         0      0    1.0               1        0.707107  264.0  15.0   1

   Ny  Nz   ShadedPAR   SunlitPAR  ShadedArea  SunlitArea      Area  \
0   1   1  380.635895  473.640411    0.010761    0.696346  0.707107

         PARa  Intercepted  Transmitted
0  472.225067     0.667827     8.742233
```

Set of triangles

For this second example, we will generate a set of random 3D triangles. A function is already implemented in the package.

[8]:
```python
from lightvegemanager.trianglesmesh import random_triangle_generator
```

[9]:
```python
nb_triangles = 5000
spheresize = (10., 2.) # vertices of the triangles are on the sphere surface
triangles = []
for i in range(nb_triangles):
    triangles.append(random_triangle_generator(spheresize=spheresize))
```

CARIBU

We repeat the same steps as with one triangle

[10]:
```
# initialize the instance
lighting = LightVegeManager(lightmodel="caribu")

# build the scene
lighting.build(geometry=triangles)

# visualisation
SceneWidget(lighting.plantGL_nolight(),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[10]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'pDz6JofPFF1NBjazXszznt9TX', 'data': b'x\
↪xda\x8c]\x07|M\xd9\xf6\x...
```

[14]:
```
# compute the lighting
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)

# print the outputs
print(lighting.triangles_outputs)
```

```
      Day  Hour  Triangle  Organ  VegetationType       Area    par Eabs  \
0     264    15         0      0               0  78.834450  139.160806
1     264    15         1      0               0  47.966120    0.886467
2     264    15         2      0               0  13.309912   27.693806
3     264    15         3      0               0  52.267733    2.489935
4     264    15         4      0               0  12.524847    0.198283
...   ...   ...       ...    ...             ...        ...         ...
4995  264    15      4995      0               0  71.155347   17.388605
4996  264    15      4996      0               0  15.119594   67.661091
4997  264    15      4997      0               0  12.947871   57.176758
4998  264    15      4998      0               0  27.825938   28.413648
4999  264    15      4999      0               0  25.764540    2.044766

          par Ei
0     163.718595
1       1.042902
2      32.580948
3       2.929335
4       0.233274
...          ...
4995   20.457182
4996   79.601283
4997   67.266774
4998   33.427821
```

(continues on next page)

(continued from previous page)

```
4999    2.405607

[5000 rows x 8 columns]
```

[13]:
```
SceneWidget(lighting.plantGL_light(),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[13]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'OGd2X5hD1gviccEoC1jTR7rLb', 'data': b'x\
↪xda\x8c}\x07X\x15\xd7\xf...
```

RATP

Now, we will set the voxels size. It needs to be specified in a dict which stores all RATP parameters. Here, you need to precise the length on each axis of one voxel.

[9]:
```
ratp_parameters = { "voxel size": [20.] * 3 }
```

Then, the dict is an argument in the instance creation.

[10]:
```
# initialize the instance
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)

# build the scene
lighting.build(geometry=triangles)

# visualisation
SceneWidget(lighting.plantGL_nolight(printtriangles=True, printvoxels=True),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[10]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'llGiOBzghQcYBBjj3iUAJ4Iks', 'data': b'x\
↪xda\x8c}\x07|O\xd9\xd6v\...
```

[11]:
```
# compute the lighting
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)

# print the outputs
print(lighting.voxels_outputs)
```

```
     VegetationType    Day  Hour  Voxel  Nx  Ny  Nz   ShadedPAR   SunlitPAR  \
0               1.0  264.0  15.0    1.0   3   3   5   40.732689  136.623886
1               1.0  264.0  15.0    2.0   5   6   6   28.347637  124.238831
```

(continues on next page)

(continued from previous page)

```
2                1.0  264.0  15.0    3.0  1   4   3  137.707138  233.598343
3                1.0  264.0  15.0    4.0  3   3   2  183.874008  279.765198
4                1.0  264.0  15.0    5.0  2   6   5   41.833862  137.725067
..               ...    ...   ...    ... ..  ..  ..         ...         ...
213              1.0  264.0  15.0  214.0  4   2   4   63.669552  159.560745
214              1.0  264.0  15.0  215.0  1   7   6  146.115402  242.006592
215              1.0  264.0  15.0  216.0  6   3   7   82.600769  178.491959
216              1.0  264.0  15.0  217.0  1   7   2  339.311615  435.202850
217              1.0  264.0  15.0  218.0  4   5   1  375.165588  471.056763

      ShadedArea  SunlitArea         Area        PARa  Intercepted  \
0    1024.282715   10.943428  1035.226196   41.746357    86.433853
1    1377.175171  283.350006  1660.525146   44.710396   148.485474
2     172.634308  291.246674   463.880981  197.912231   183.615433
3     486.705994  360.075684   846.781677  224.649658   380.458435
4    1375.770874   41.413651  1417.184570   44.636040   126.515022
..           ...         ...          ...         ...          ...
213   626.312683  157.908005   784.220703   82.977875   130.145935
214    57.020470   86.957054   143.977524  204.030106    58.751499
215   197.366257  156.226395   353.592651  124.967972    88.375519
216     4.840422   17.089012    21.929434  414.037079    18.159195
217     0.815029   39.664459    40.479488  469.126038    37.979965

     Transmitted
0      31.874031
1      18.599306
2     119.887787
3     101.195755
4      33.631508
..           ...
213    52.329762
214   128.099014
215    82.136887
216   328.422760
217   385.791229

[218 rows x 15 columns]
```

[23]:
```
# print the outputs
print(lighting.triangles_outputs)
```

```
      Triangle  Organ  Voxel  VegetationType  primitive_area    Day  Hour  Nx  \
0            0      0    1.0               1        6.008933  264.0  15.0   5
49           1      0    2.0               1       64.911050  264.0  15.0   4
77           2      0    3.0               1       48.494517  264.0  15.0   3
84           3      0    4.0               1        5.762879  264.0  15.0   1
91           4      0    5.0               1       21.321846  264.0  15.0   3
...        ...    ...    ...             ...             ...    ...   ...  ..
586       4995      0   21.0               1       21.090689  264.0  15.0   5
3641      4996      0  136.0               1       32.577000  264.0  15.0   4
1051      4997      0   37.0               1       76.699258  264.0  15.0   6
1822      4998      0   63.0               1        1.464895  264.0  15.0   2
```

(continues on next page)

(continued from previous page)

```
3443      4999      0   129.0             1       51.065793   264.0   15.0    5

      Ny   Nz    ShadedPAR    SunlitPAR    ShadedArea   SunlitArea          Area   \
0      6    3    48.783260  144.457764  1669.359253  307.066376  1976.425659
49     2    6    50.222652  145.897171   639.340942  467.153961  1106.494873
77     7    7    76.529640  172.204147   131.880585   49.266487   181.147064
84     7    4   162.029709  257.704224   114.480240  108.982895   223.463135
91     5    5    36.165016  131.839523  1061.264648    3.100358  1064.364990
...   ..   ..          ...         ...          ...         ...          ...
586    4    6    28.958939  124.633453   982.723572  278.425110  1261.148682
3641   4    4    20.419191  116.093712  1379.531006   34.760368  1414.291382
1051   6    5    68.352638  164.027176  1125.141846    1.110655  1126.252441
1822   5    5    39.313770  134.988281  1225.665894   14.276536  1239.942383
3443   2    6    60.735207  156.409714   431.796021  383.754333   815.550354

            PARa  Intercepted  Transmitted
0      63.647678   251.589828    31.452011
49     90.615723   200.531677    35.146641
77    102.550201    37.153339    72.656891
84    208.690125    93.269104   155.314255
91     36.443703    77.578804    28.307266
...          ...          ...          ...
586    50.081097   126.319427    21.615057
3641   22.770676    64.408737    13.644242
1051   68.446991   154.177170    60.248528
1822   40.415356   100.225418    28.215555
3443  105.754509   172.496262    47.396954

[5000 rows x 18 columns]
```

[13]:
```python
# visualisation
SceneWidget(lighting.plantGL_light(printtriangles=True, printvoxels=False),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[13]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'OaYtClMfKTsXFShXk1ESHgnYz', 'data': b'x\
↪xda\x84]\x07xMY\xd7\x8e\...
```

[14]:
```python
# visualisation
SceneWidget(lighting.plantGL_light(printtriangles=False, printvoxels=True),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[14]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'jWP5jHrIFv0YocDTC7u66AES2', 'data': b'x\
↪xda\x95]K\x8f\xa6\xc7U\x...
```

2.6.2 Environment parameters: ways to set the simulation environment

Content

- Coordinates and other boolean options
- sky
  - turtle of 46 directions
  - file
  - custom number of sky directions

Introduction

Environment parameters are stored in a dict and transferred as an input argument for a LightVegeManager instance. It defines all static parameters during a simulation, such the sky type, radiative options etc…

```
environment = {
                "coordinates" : [latitude, longitude, timezone] ,

                "sky" : "turtle46" ,
                "sky" : ["file", filepath] ,
                "sky" : [nb_azimut, nb_zenith, "soc" or "uoc"] ,

                "direct" : bool, # sun radiations
                "diffus" : bool, # sky radiations
                "reflected" : bool, # reflected radiation in the canopy
                "infinite" : bool, # infinitisation of the scene
                }
```

[1]:
```
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
```

As a geometric example, we will use a random set of 3D triangles

[2]:
```
from lightvegemanager.trianglesmesh import random_triangle_generator

nb_triangles = 50
spheresize = (10., 2.) # vertices of triangles are the sphere surface
triangles = []
for i in range(nb_triangles):
    triangles.append(random_triangle_generator(spheresize=spheresize))
```

First options

- "coordinates": sets the coordinates of the simulation, this matters if you want direct radiations
- "infinite": activates infinite reproduction
- "reflected": activates the reflections in the scene
- "direct": activates the direct radiations (sun)
- "diffuse": activates the diffuse radiations (sky)

An example of use

[3]:
```
longitude = 2.
latitude = 46.
timezone = 1
coordinates = [latitude, longitude, timezone]
infinite = False
reflected = False
direct = False
diffuse = True

environment = {
                "coordinates": coordinates ,
                "infinite": infinite,
                "reflected": reflected,
                "direct": direct,
                "diffuse": diffuse
                }
```

[4]:
```
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=triangles)

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.triangles_outputs)
```

```
    Day  Hour  Triangle  Organ  VegetationType        Area  par Eabs  \
0   264    15         0      0               0   99.480976  313.1030
1   264    15         1      0               0  140.415211  318.7210
2   264    15         2      0               0   20.519232  384.2305
3   264    15         3      0               0   23.926310  372.2110
4   264    15         4      0               0    5.986450  414.9295
5   264    15         5      0               0   10.845583  389.0330
6   264    15         6      0               0    0.882299  385.6705
7   264    15         7      0               0   32.812203  350.8225
8   264    15         8      0               0   36.321775  375.5740
9   264    15         9      0               0    5.071856  409.3300
10  264    15        10      0               0   52.628089  366.5175
11  264    15        11      0               0  103.885213  365.5105
12  264    15        12      0               0   65.457326  362.9340
13  264    15        13      0               0    0.567605  371.0460
14  264    15        14      0               0    7.122030  398.2765
```

(continues on next page)

(continued from previous page)

```
15  264    15         15      0              0    5.181860  342.8800
16  264    15         16      0              0  271.820976  384.4815
17  264    15         17      0              0   91.585066  382.0960
18  264    15         18      0              0   36.315518  352.2335
19  264    15         19      0              0   31.149412  375.1325
20  264    15         20      0              0   35.587854  371.4830
21  264    15         21      0              0   14.172730  387.1420
22  264    15         22      0              0   31.918145  306.4970
23  264    15         23      0              0   76.036115  360.8220
24  264    15         24      0              0   13.932378  348.0675
25  264    15         25      0              0   55.601914  391.2430
26  264    15         26      0              0   84.448667  373.2410
27  264    15         27      0              0   90.626852  390.3190
28  264    15         28      0              0   19.211308  305.5760
29  264    15         29      0              0    7.412773  342.4710
30  264    15         30      0              0   50.867131  386.5955
31  264    15         31      0              0   22.183195  356.0750
32  264    15         32      0              0    5.146020  341.1475
33  264    15         33      0              0   14.648915  367.1705
34  264    15         34      0              0   64.285771  355.4715
35  264    15         35      0              0   15.752372  373.9875
36  264    15         36      0              0   34.818585  365.3500
37  264    15         37      0              0   61.748544  363.2435
38  264    15         38      0              0  102.627411  413.6665
39  264    15         39      0              0    6.460148  330.7685
40  264    15         40      0              0   45.802731  373.1850
41  264    15         41      0              0   21.008426  356.8730
42  264    15         42      0              0    9.276760  334.7805
43  264    15         43      0              0   28.316439  353.1385
44  264    15         44      0              0   36.758141  335.3380
45  264    15         45      0              0  121.996376  346.0755
46  264    15         46      0              0   69.352270  351.8705
47  264    15         47      0              0   71.266675  365.3110
48  264    15         48      0              0   81.346766  401.6110
49  264    15         49      0              0   13.311873  406.2700

        par Ei
0   368.356471
1   374.965882
2   452.035882
3   437.895294
4   488.152353
5   457.685882
6   453.730000
7   412.732353
8   441.851765
9   481.564706
10  431.197059
11  430.012353
12  426.981176
13  436.524706
14  468.560588
```

(continues on next page)

(continued from previous page)

```
15   403.388235
16   452.331176
17   449.524706
18   414.392353
19   441.332353
20   437.038824
21   455.461176
22   360.584706
23   424.496471
24   409.491176
25   460.285882
26   439.107059
27   459.198824
28   359.501176
29   402.907059
30   454.818235
31   418.911765
32   401.350000
33   431.965294
34   418.201765
35   439.985294
36   429.823529
37   427.345294
38   486.666471
39   389.139412
40   439.041176
41   419.850588
42   393.859412
43   415.457059
44   394.515294
45   407.147647
46   413.965294
47   429.777647
48   472.483529
49   477.964706
```

Different ways to set a sky

Turtle46

This options creates a sky in a turtle of 46 directions. It is the default sky in the tool.

[5]:
```
sky = "turtle46"
environment.update({"sky": sky})
```

[6]:
```
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=triangles)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.triangles_outputs)
```

```
    Day  Hour  Triangle  Organ  VegetationType        Area  par Eabs  \
0   264    15         0      0               0   99.480976  313.1030
1   264    15         1      0               0  140.415211  318.7210
2   264    15         2      0               0   20.519232  384.2305
3   264    15         3      0               0   23.926310  372.2110
4   264    15         4      0               0    5.986450  414.9295
5   264    15         5      0               0   10.845583  389.0330
6   264    15         6      0               0    0.882299  385.6705
7   264    15         7      0               0   32.812203  350.8225
8   264    15         8      0               0   36.321775  375.5740
9   264    15         9      0               0    5.071856  409.3300
10  264    15        10      0               0   52.628089  366.5175
11  264    15        11      0               0  103.885213  365.5105
12  264    15        12      0               0   65.457326  362.9340
13  264    15        13      0               0    0.567605  371.0460
14  264    15        14      0               0    7.122030  398.2765
15  264    15        15      0               0    5.181860  342.8800
16  264    15        16      0               0  271.820976  384.4815
17  264    15        17      0               0   91.585066  382.0960
18  264    15        18      0               0   36.315518  352.2335
19  264    15        19      0               0   31.149412  375.1325
20  264    15        20      0               0   35.587854  371.4830
21  264    15        21      0               0   14.172730  387.1420
22  264    15        22      0               0   31.918145  306.4970
23  264    15        23      0               0   76.036115  360.8220
24  264    15        24      0               0   13.932378  348.0675
25  264    15        25      0               0   55.601914  391.2430
26  264    15        26      0               0   84.448667  373.2410
27  264    15        27      0               0   90.626852  390.3190
28  264    15        28      0               0   19.211308  305.5760
29  264    15        29      0               0    7.412773  342.4710
30  264    15        30      0               0   50.867131  386.5955
31  264    15        31      0               0   22.183195  356.0750
32  264    15        32      0               0    5.146020  341.1475
33  264    15        33      0               0   14.648915  367.1705
34  264    15        34      0               0   64.285771  355.4715
35  264    15        35      0               0   15.752372  373.9875
36  264    15        36      0               0   34.818585  365.3500
37  264    15        37      0               0   61.748544  363.2435
38  264    15        38      0               0  102.627411  413.6665
39  264    15        39      0               0    6.460148  330.7685
40  264    15        40      0               0   45.802731  373.1850
41  264    15        41      0               0   21.008426  356.8730
42  264    15        42      0               0    9.276760  334.7805
43  264    15        43      0               0   28.316439  353.1385
44  264    15        44      0               0   36.758141  335.3380
45  264    15        45      0               0  121.996376  346.0755
46  264    15        46      0               0   69.352270  351.8705
47  264    15        47      0               0   71.266675  365.3110
48  264    15        48      0               0   81.346766  401.6110
49  264    15        49      0               0   13.311873  406.2700

        par Ei
```

(continues on next page)

(continued from previous page)

```
0   368.356471
1   374.965882
2   452.035882
3   437.895294
4   488.152353
5   457.685882
6   453.730000
7   412.732353
8   441.851765
9   481.564706
10  431.197059
11  430.012353
12  426.981176
13  436.524706
14  468.560588
15  403.388235
16  452.331176
17  449.524706
18  414.392353
19  441.332353
20  437.038824
21  455.461176
22  360.584706
23  424.496471
24  409.491176
25  460.285882
26  439.107059
27  459.198824
28  359.501176
29  402.907059
30  454.818235
31  418.911765
32  401.350000
33  431.965294
34  418.201765
35  439.985294
36  429.823529
37  427.345294
38  486.666471
39  389.139412
40  439.041176
41  419.850588
42  393.859412
43  415.457059
44  394.515294
45  407.147647
46  413.965294
47  429.777647
48  472.483529
49  477.964706
```

File

You can set sky directions in a file. It needs azimut, zenit, weight and solid angle of each direction.

[7]:
```
import os
datafile = os.path.join(os.path.join(os.path.dirname(os.path.abspath("")), "data"), "sky_
↪5.data")
datafile
```

[7]:
```
'C:\\Users\\mwoussen\\cdd\\codes\\dev\\lightvegemanager\\data\\sky_5.data'
```

[8]:
```
sky = datafile
environment.update({"sky": sky})
```

[9]:
```
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=triangles)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.triangles_outputs)
```

```
    Day  Hour  Triangle  Organ  VegetationType        Area  par Eabs  \
0   264    15         0      0               0   99.480976  321.8920
1   264    15         1      0               0  140.415211  410.9825
2   264    15         2      0               0   20.519232  349.7725
3   264    15         3      0               0   23.926310  390.1275
4   264    15         4      0               0    5.986450  404.9985
5   264    15         5      0               0   10.845583  405.4615
6   264    15         6      0               0    0.882299  388.1070
7   264    15         7      0               0   32.812203  375.9985
8   264    15         8      0               0   36.321775  378.9715
9   264    15         9      0               0    5.071856  393.7835
10  264    15        10      0               0   52.628089  391.1865
11  264    15        11      0               0  103.885213  396.1285
12  264    15        12      0               0   65.457326  381.3970
13  264    15        13      0               0    0.567605  382.6855
14  264    15        14      0               0    7.122030  392.8865
15  264    15        15      0               0    5.181860  336.5425
16  264    15        16      0               0  271.820976  393.6545
17  264    15        17      0               0   91.585066  357.0890
18  264    15        18      0               0   36.315518  383.2600
19  264    15        19      0               0   31.149412  361.9780
20  264    15        20      0               0   35.587854  376.8490
21  264    15        21      0               0   14.172730  380.3260
22  264    15        22      0               0   31.918145  361.1125
23  264    15        23      0               0   76.036115  399.0185
24  264    15        24      0               0   13.932378  343.3480
25  264    15        25      0               0   55.601914  411.6490
26  264    15        26      0               0   84.448667  396.3155
27  264    15        27      0               0   90.626852  376.1960
28  264    15        28      0               0   19.211308  283.1020
29  264    15        29      0               0    7.412773  384.0860
30  264    15        30      0               0   50.867131  357.6440
31  264    15        31      0               0   22.183195  383.9450
32  264    15        32      0               0    5.146020  430.5005
33  264    15        33      0               0   14.648915  416.5190
```

(continues on next page)

(continued from previous page)

```
34  264    15        34      0                0   64.285771  414.2600
35  264    15        35      0                0   15.752372  263.4695
36  264    15        36      0                0   34.818585  389.2345
37  264    15        37      0                0   61.748544  336.5750
38  264    15        38      0                0  102.627411  402.2385
39  264    15        39      0                0    6.460148  373.0750
40  264    15        40      0                0   45.802731  397.2810
41  264    15        41      0                0   21.008426  417.2495
42  264    15        42      0                0    9.276760  404.4165
43  264    15        43      0                0   28.316439  386.4300
44  264    15        44      0                0   36.758141  411.7950
45  264    15        45      0                0  121.996376  379.8200
46  264    15        46      0                0   69.352270  327.2225
47  264    15        47      0                0   71.266675  389.3355
48  264    15        48      0                0   81.346766  381.5730
49  264    15        49      0                0   13.311873  383.4205

        par Ei
0   378.696471
1   483.508824
2   411.497059
3   458.973529
4   476.468824
5   477.013529
6   456.596471
7   442.351176
8   445.848824
9   463.274706
10  460.219412
11  466.033529
12  448.702353
13  450.218235
14  462.219412
15  395.932353
16  463.122941
17  420.104706
18  450.894118
19  425.856471
20  443.351765
21  447.442353
22  424.838235
23  469.433529
24  403.938824
25  484.292941
26  466.253529
27  442.583529
28  333.061176
29  451.865882
30  420.757647
31  451.700000
32  506.471176
33  490.022353
```

(continues on next page)

(continued from previous page)

```
34   487.364706
35   309.964118
36   457.922941
37   395.970588
38   473.221765
39   438.911765
40   467.389412
41   490.881765
42   475.784118
43   454.623529
44   484.464706
45   446.847059
46   384.967647
47   458.041765
48   448.909412
49   451.082941
```

Custom number of directions

You can directly precise the number of directions for each spherical axis.

[10]:
```
nazimuts = 5
nzenits = 5
skytype = "soc"
sky = [nazimuts, nzenits, skytype]
environment.update({"sky": sky})
```

[11]:
```
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=triangles)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.triangles_outputs)
```

```
    Day  Hour  Triangle  Organ  VegetationType        Area  par Eabs  \
0   264    15         0      0               0   99.480976  312.9400
1   264    15         1      0               0  140.415211  307.4715
2   264    15         2      0               0   20.519232  376.8230
3   264    15         3      0               0   23.926310  380.4820
4   264    15         4      0               0    5.986450  400.4520
5   264    15         5      0               0   10.845583  410.6180
6   264    15         6      0               0    0.882299  382.5890
7   264    15         7      0               0   32.812203  342.8405
8   264    15         8      0               0   36.321775  378.7905
9   264    15         9      0               0    5.071856  409.1425
10  264    15        10      0               0   52.628089  358.2720
11  264    15        11      0               0  103.885213  370.8735
12  264    15        12      0               0   65.457326  373.1205
13  264    15        13      0               0    0.567605  372.0525
14  264    15        14      0               0    7.122030  408.1645
15  264    15        15      0               0    5.181860  320.8560
16  264    15        16      0               0  271.820976  382.6820
17  264    15        17      0               0   91.585066  340.6655
```

(continues on next page)

(continued from previous page)

```
18  264   15      18     0               0   36.315518  339.7660
19  264   15      19     0               0   31.149412  385.4065
20  264   15      20     0               0   35.587854  357.5670
21  264   15      21     0               0   14.172730  393.8765
22  264   15      22     0               0   31.918145  307.1930
23  264   15      23     0               0   76.036115  352.0500
24  264   15      24     0               0   13.932378  341.7345
25  264   15      25     0               0   55.601914  387.1755
26  264   15      26     0               0   84.448667  375.2625
27  264   15      27     0               0   90.626852  391.0160
28  264   15      28     0               0   19.211308  285.6480
29  264   15      29     0               0    7.412773  359.1335
30  264   15      30     0               0   50.867131  393.6610
31  264   15      31     0               0   22.183195  343.9185
32  264   15      32     0               0    5.146020  315.2815
33  264   15      33     0               0   14.648915  357.9085
34  264   15      34     0               0   64.285771  357.0725
35  264   15      35     0               0   15.752372  371.3890
36  264   15      36     0               0   34.818585  370.8550
37  264   15      37     0               0   61.748544  349.6305
38  264   15      38     0               0  102.627411  413.2585
39  264   15      39     0               0    6.460148  317.7325
40  264   15      40     0               0   45.802731  377.5860
41  264   15      41     0               0   21.008426  335.8295
42  264   15      42     0               0    9.276760  346.0530
43  264   15      43     0               0   28.316439  370.1290
44  264   15      44     0               0   36.758141  333.0075
45  264   15      45     0               0  121.996376  362.9850
46  264   15      46     0               0   69.352270  347.6555
47  264   15      47     0               0   71.266675  371.9740
48  264   15      48     0               0   81.346766  397.0950
49  264   15      49     0               0   13.311873  406.0575

        par Ei
0   368.164706
1   361.731176
2   443.321176
3   447.625882
4   471.120000
5   483.080000
6   450.104706
7   403.341765
8   445.635882
9   481.344118
10  421.496471
11  436.321765
12  438.965294
13  437.708824
14  480.193529
15  377.477647
16  450.214118
17  400.782941
```

(continues on next page)

(continued from previous page)

```
18   399.724706
19   453.419412
20   420.667059
21   463.384118
22   361.403529
23   414.176471
24   402.040588
25   455.500588
26   441.485294
27   460.018824
28   336.056471
29   422.510000
30   463.130588
31   404.610000
32   370.919412
33   421.068824
34   420.085294
35   436.928235
36   436.300000
37   411.330000
38   486.186471
39   373.802941
40   444.218824
41   395.093529
42   407.121176
43   435.445882
44   391.773529
45   427.041176
46   409.006471
47   437.616471
48   467.170588
49   477.714706
```

And RATP ?

All the skies defined above are available with RATP as the light model

[12]:
```
lighting = LightVegeManager(lightmodel="ratp", environment=environment)
lighting.build(geometry=triangles)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.triangles_outputs)
```

```
    Triangle  Organ  Voxel  VegetationType  primitive_area    Day  Hour  Nx  \
0          0      0    1.0               1       99.480976  264.0  15.0   2
7          1      0    2.0               1      140.415211  264.0  15.0   2
14         2      0    3.0               1       20.519232  264.0  15.0   1
25         3      0    4.0               1       23.926310  264.0  15.0   1
26         4      0    4.0               1        5.986450  264.0  15.0   1
15         5      0    3.0               1       10.845583  264.0  15.0   1
16         6      0    3.0               1        0.882299  264.0  15.0   1
29         7      0    5.0               1       32.812203  264.0  15.0   1
```

(continues on next page)

(continued from previous page)

```
27         8      0    4.0               1       36.321775  264.0  15.0   1
1          9      0    1.0               1        5.071856  264.0  15.0   2
34        10      0    6.0               1       52.628089  264.0  15.0   2
2         11      0    1.0               1      103.885213  264.0  15.0   2
17        12      0    3.0               1       65.457326  264.0  15.0   1
18        13      0    3.0               1        0.567605  264.0  15.0   1
19        14      0    3.0               1        7.122030  264.0  15.0   1
39        15      0    7.0               1        5.181860  264.0  15.0   1
20        16      0    3.0               1      271.820976  264.0  15.0   1
21        17      0    3.0               1       91.585066  264.0  15.0   1
3         18      0    1.0               1       36.315518  264.0  15.0   2
42        19      0    8.0               1       31.149412  264.0  15.0   2
8         20      0    2.0               1       35.587854  264.0  15.0   2
22        21      0    3.0               1       14.172730  264.0  15.0   1
40        22      0    7.0               1       31.918145  264.0  15.0   1
43        23      0    8.0               1       76.036115  264.0  15.0   2
35        24      0    6.0               1       13.932378  264.0  15.0   2
44        25      0    8.0               1       55.601914  264.0  15.0   2
36        26      0    6.0               1       84.448667  264.0  15.0   2
30        27      0    5.0               1       90.626852  264.0  15.0   1
4         28      0    1.0               1       19.211308  264.0  15.0   2
45        29      0    8.0               1        7.412773  264.0  15.0   2
9         30      0    2.0               1       50.867131  264.0  15.0   2
46        31      0    8.0               1       22.183195  264.0  15.0   2
47        32      0    8.0               1        5.146020  264.0  15.0   2
31        33      0    5.0               1       14.648915  264.0  15.0   1
28        34      0    4.0               1       64.285771  264.0  15.0   1
48        35      0    8.0               1       15.752372  264.0  15.0   2
32        36      0    5.0               1       34.818585  264.0  15.0   1
10        37      0    2.0               1       61.748544  264.0  15.0   2
5         38      0    1.0               1      102.627411  264.0  15.0   2
11        39      0    2.0               1        6.460148  264.0  15.0   2
41        40      0    7.0               1       45.802731  264.0  15.0   1
6         41      0    1.0               1       21.008426  264.0  15.0   2
12        42      0    2.0               1        9.276760  264.0  15.0   2
23        43      0    3.0               1       28.316439  264.0  15.0   1
49        44      0    8.0               1       36.758141  264.0  15.0   2
24        45      0    3.0               1      121.996376  264.0  15.0   1
13        46      0    2.0               1       69.352270  264.0  15.0   2
37        47      0    6.0               1       71.266675  264.0  15.0   2
38        48      0    6.0               1       81.346766  264.0  15.0   2
33        49      0    5.0               1       13.311873  264.0  15.0   1

    Ny  Nz   ShadedPAR   SunlitPAR  ShadedArea  SunlitArea        Area  \
0    2   1  514.910706  514.910706    8.961945  378.638763  387.600708
7    1   2  508.419556  508.419556   10.553925  363.153992  373.707916
14   2   2  503.838837  503.838837   19.746033  613.539612  633.285645
25   1   1  520.405823  520.405823    1.771942  128.748367  130.520309
26   1   1  520.405823  520.405823    1.771942  128.748367  130.520309
15   2   2  503.838837  503.838837   19.746033  613.539612  633.285645
16   2   2  503.838837  503.838837   19.746033  613.539612  633.285645
29   2   1  519.076111  519.076111    2.660919  183.557510  186.218430
```

(continues on next page)

(continued from previous page)

```
27   1   1  520.405823  520.405823   1.771942  128.748367  130.520309
1    2   1  514.910706  514.910706   8.961945  378.638763  387.600708
34   1   1  516.687439  516.687439   6.621002  297.001587  303.622589
2    2   1  514.910706  514.910706   8.961945  378.638763  387.600708
17   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
18   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
19   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
39   1   2  514.584290  514.584290   1.922020   80.980721   82.902740
20   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
21   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
3    2   1  514.910706  514.910706   8.961945  378.638763  387.600708
42   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
8    1   2  508.419556  508.419556  10.553925  363.153992  373.707916
22   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
40   1   2  514.584290  514.584290   1.922020   80.980721   82.902740
43   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
35   1   1  516.687439  516.687439   6.621002  297.001587  303.622589
44   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
36   1   1  516.687439  516.687439   6.621002  297.001587  303.622589
30   2   1  519.076111  519.076111   2.660919  183.557510  186.218430
4    2   1  514.910706  514.910706   8.961945  378.638763  387.600708
45   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
9    1   2  508.419556  508.419556  10.553925  363.153992  373.707916
46   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
47   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
31   2   1  519.076111  519.076111   2.660919  183.557510  186.218430
28   1   1  520.405823  520.405823   1.771942  128.748367  130.520309
48   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
32   2   1  519.076111  519.076111   2.660919  183.557510  186.218430
10   1   2  508.419556  508.419556  10.553925  363.153992  373.707916
5    2   1  514.910706  514.910706   8.961945  378.638763  387.600708
11   1   2  508.419556  508.419556  10.553925  363.153992  373.707916
41   1   2  514.584290  514.584290   1.922020   80.980721   82.902740
6    2   1  514.910706  514.910706   8.961945  378.638763  387.600708
12   1   2  508.419556  508.419556  10.553925  363.153992  373.707916
23   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
49   2   2  508.459503  508.459503   6.290421  243.749527  250.039948
24   2   2  503.838837  503.838837  19.746033  613.539612  633.285645
13   1   2  508.419556  508.419556  10.553925  363.153992  373.707916
37   1   1  516.687439  516.687439   6.621002  297.001587  303.622589
38   1   1  516.687439  516.687439   6.621002  297.001587  303.622589
33   2   1  519.076111  519.076111   2.660919  183.557510  186.218430

          PARa  Intercepted  Transmitted
0   514.910706   399.159515  4677.770020
7   508.419586   380.000824  4636.436523
14  503.838837   638.147766  4564.685547
25  520.405823   135.847061  4753.783691
26  520.405823   135.847061  4753.783691
15  503.838837   638.147766  4564.685547
16  503.838837   638.147766  4564.685547
29  519.076111   193.323074  4734.602051
```

(continues on next page)

(continued from previous page)

```
27  520.405823   135.847061  4753.783691
1   514.910706   399.159515  4677.770020
34  516.687439   313.755951  4702.907227
2   514.910706   399.159515  4677.770020
17  503.838837   638.147766  4564.685547
18  503.838837   638.147766  4564.685547
19  503.838837   638.147766  4564.685547
39  514.584290    85.320900  4692.383789
20  503.838837   638.147766  4564.685547
21  503.838837   638.147766  4564.685547
3   514.910706   399.159515  4677.770020
42  508.459503   254.270370  4635.730469
8   508.419586   380.000824  4636.436523
22  503.838837   638.147766  4564.685547
40  514.584290    85.320900  4692.383789
43  508.459503   254.270370  4635.730469
35  516.687439   313.755951  4702.907227
44  508.459503   254.270370  4635.730469
36  516.687439   313.755951  4702.907227
30  519.076111   193.323074  4734.602051
4   514.910706   399.159515  4677.770020
45  508.459503   254.270370  4635.730469
9   508.419586   380.000824  4636.436523
46  508.459503   254.270370  4635.730469
47  508.459503   254.270370  4635.730469
31  519.076111   193.323074  4734.602051
28  520.405823   135.847061  4753.783691
48  508.459503   254.270370  4635.730469
32  519.076111   193.323074  4734.602051
10  508.419586   380.000824  4636.436523
5   514.910706   399.159515  4677.770020
11  508.419586   380.000824  4636.436523
41  514.584290    85.320900  4692.383789
6   514.910706   399.159515  4677.770020
12  508.419586   380.000824  4636.436523
23  503.838837   638.147766  4564.685547
49  508.459503   254.270370  4635.730469
24  503.838837   638.147766  4564.685547
13  508.419586   380.000824  4636.436523
37  516.687439   313.755951  4702.907227
38  516.687439   313.755951  4702.907227
33  519.076111   193.323074  4734.602051
```

2.6.3 Geometric inputs

Content

- input formats
  - Dict of triangles
  - PlantGL scene
  - VGX file
  - Voxels grid
  - MTG object from adelwheat
  - RATP inputs
  - RiRi5 inputs
- organization levels: species and organs
- stems management
- geometric transformations

Introduction

The main purpose of this tool was to merge several geometric scenes in various formats and apply a radiative modelling on it. Here, we will precise the different possiblities for manipulating geometry.

[1]:
```
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
```

[2]:
```
# Environment parameters for this notebook
longitude = 2.
latitude = 46.
timezone = 1
coordinates = [latitude, longitude, timezone]
infinite = False
reflected = False
direct = False
diffuse = True
sky = "turtle46"

environment = {
                "coordinates": coordinates ,
                "infinite": infinite,
                "reflected": reflected,
                "direct": direct,
                "diffuse": diffuse,
                "sky": sky
                }
```

Inputs Formats

In this section, we won't present single triangle and list of triangles as input geometry, as they were present in the `tool_basics.ipynb` notebook. Also, these two formats can be direct inputs of the `geometry` argument of the `build` method, unlike the following formats which needs to be included in a list, such as:

```
geometry = { "scenes": [scene1, scene2, ...] }
```

dict of triangles

A mesh of triangles can be represented as a dict, where each key is an organ ID and its value, a list of triangles belonging to the organ. A triangle is a list of a 3 vertices represented by `(x,y,z)`points.

In this example, we will generate random 3D triangles for 3 differents organs.

[12]:
```
from lightvegemanager.trianglesmesh import random_triangle_generator

spheresize = (10., 2.)
organized_triangles = {
    111: [random_triangle_generator(spheresize=spheresize, worldsize=(0,20)) for i in␣
↪range(20)],
    222: [random_triangle_generator(spheresize=spheresize, worldsize=(20,50)) for i in␣
↪range(30)],
    333: [random_triangle_generator(spheresize=spheresize, worldsize=(50,100)) for i in␣
↪range(10)],
}
```

Input geometry looks like:

[4]:
```
geometry = {
    "scenes": [organized_triangles]
}
```

Then, we compute lighting and run a visualization of the scene.

Note: `elements_outputs` method will return a Dataframe where results are integrated on each organ.

[5]:
```
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=geometry)

# compute the lighting
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

```
   Day  Hour  Organ  VegetationType        Area    par Eabs      par Ei
0  264    15    111               0  812.660080  254.783023  299.744733
1  264    15    222               0  906.791550  326.191765  383.755017
2  264    15    333               0  456.917637  368.655503  433.712356
```

[5]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'BwxHSpLTosrUCKkss9pQniZu1', 'data': b'x\
↪xda\x85\x9a\t\\U\xd5\xf6...
```

PlantGL scene

A plantGL Scene is a list of plantGL Shape which can be considered as organ. The ID of the plantGL Shape are stored as organs ID.

[7]:
```
import openalea.plantgl.all as pgl

pgl_scene = pgl.Scene([pgl.Shape(pgl.Box(), pgl.Material(), 888),
                       pgl.Shape(pgl.Translated((0,0,1), pgl.Cylinder()), pgl.
↪Material(), 999)])
```

[8]:
```
geometry = {
    "scenes": [pgl_scene]
}
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 5,
            axes_helper=True)
```

```
   Day  Hour  Organ  VegetationType      Area    par Eabs      par Ei
0  264    15    888               0  6.000000  159.643042  187.815343
1  264    15    999               0  4.475681  185.076017  217.736490
```

[8]:
```
SceneWidget(axes_helper=True, scenes=[{'id': '8kU7diEsaj9G3BXsKdlxCbLNN', 'data': b'x\
↪xda\x95\x99\xddn\x1bE\x1...
```

VGX file

The tool can read a VGX file as an input entry. It extracts triangles which are considered as leaves, following its colors, if Red != 42 . All triangles are stored in the same organ, where its ID is set to 0.

[5]:
```
import os

vgx_path = os.path.join(os.path.dirname(os.path.abspath("")), "data", "NICatObs1P2.vgx")
vgx_path
```

[5]:
```
'C:\\Users\\mwoussen\\cdd\\codes\\dev\\lightvegemanager\\data\\NICatObs1P2.vgx'
```

[7]:
```
geometry = {
    "scenes": [vgx_path]
}
# compute the lighting
energy = 500.
hour = 15
day = 264
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

```
   Day  Hour  Organ  VegetationType         Area    par Eabs      par Ei
0  264    15      0               0  6062.836763  193.704509  227.887658
```

[7]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'bOBZCLMbmtxLySxt6uCQ3jfKw', 'data': b'x\
↪xda\x94\x9d\x07\x9c\x15E...
```

Grid of voxels

A voxel grid is represented as a dict of two entries: - "LA" corresponding to leaf area, is a table (`numpy.array`) of dimension $:nbsphinx-math:*text{number of species}* $\times$‘ : *nbsphinx − math : text{numberofzlayers}*‘ $\times$ ‘ : *nbsphinx − math : text{numberofxlayers}*‘ $\times$ ‘ : *nbsphinx − math : text{numberofylayers}*‘$ − `"distrib"`*correspondingtoleafangledistribution, isalistoflist, whereisenteredaleafangledistributionforeachspecy*

Grid dimensions and voxel size are set in the input parameters of RATP.

[7]:
```
import numpy

l_scene = {"LA": numpy.ones([2, 3, 4, 4]), "distrib": [[0.5, 0.5], [0.3, 0.7]]}
```

[16]:
```
geometry = {
    "scenes": [vgx_path]
}
ratp_parameters = {
    "voxel size" : [20.] * 3
}
lighting = LightVegeManager(lightmodel="ratp", environment=environment, lightmodel_
↪parameters=ratp_parameters)
lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(printtriangles=False, printvoxels=True),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
```

(continues on next page)

(continued from previous page)

```
            size_world = 100,
            axes_helper=True)
```

```
     Day  Hour  Organ  VegetationType         Area        PARa  Intercepted  \
0  264.0  15.0      0               1  6062.836763  307.604456   215.922903

   Transmitted   SunlitPAR  SunlitArea   ShadedPAR  ShadedArea
0   215.922903  307.604461  252.377915  307.604461  126.601372
```

[16]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'OgKLbEOYkRT69libXrZkXUjuC', 'data': b'x\
↪xda\x95XMh]E\x14\xbe4\x8...
```

MTG object from adelwheat

Finally, you can also give a MTG object with a "scene" property. The package adelwheat offers such objects.

[8]:
```
from alinea.adel.adel_dynamic import AdelDyn
from alinea.adel.echap_leaf import echap_leaves
INPUTS_DIRPATH = os.path.join(os.path.dirname(os.path.abspath("")), "data")
adel_wheat = AdelDyn(seed=1, scene_unit="m", leaves=echap_leaves(xy_model="Soissons_
↪byleafclass"))
g = adel_wheat.load(dir=INPUTS_DIRPATH)
```

[18]:
```
geometry = {
    "scenes": [g]
}
lighting = LightVegeManager(lightmodel="caribu", environment=environment)
lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 0.1,
            axes_helper=True)
```

```
   Day  Hour  Organ  VegetationType      Area    par Eabs      par Ei
0  264    15     19               0  0.000228  159.208612  187.304249
1  264    15     34               0  0.000013   80.953500   95.239412
2  264    15    813               0  0.000194  385.317031  453.314154
3  264    15    814               0  0.000240  367.646906  432.525772
4  264    15     51               0  0.000284  347.458387  408.774573
```

[18]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'nOBWAHxiPOAKSUmQc0cHKQkiR', 'data': b'x\
↪xda\x95\x9a{pUW\x15\xc6/...
```

RATP inputs

All the above scenes can be geometric inputs if the lightmodel argument is set to "ratp".

[21]:
```
geometry = {
    "scenes": [g]
}
lighting = LightVegeManager(lightmodel="ratp", environment=environment)
lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(printvoxels=True),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 0.1,
            axes_helper=True)
```

```
     Day  Hour  Organ  VegetationType      Area        PARa  Intercepted  \
0  264.0  15.0     19               1  0.000228  435.713379     0.000693
1  264.0  15.0     34               1  0.000013  435.713379     0.000693
2  264.0  15.0    813               1  0.000194  435.713379     0.000693
3  264.0  15.0    814               1  0.000240  441.368498     0.000288
4  264.0  15.0     51               1  0.000284  435.713379     0.000693

   Transmitted   SunlitPAR  SunlitArea   ShadedPAR  ShadedArea
0     0.000693  435.713379    0.000776  435.713379    0.000019
1     0.000693  435.713379    0.000776  435.713379    0.000019
2     0.000693  435.713379    0.000776  435.713379    0.000019
3     0.000288  441.368498    0.000322  441.368498    0.000007
4     0.000693  435.713379    0.000776  435.713379    0.000019
```

[21]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'wctv0d7i6YHequCnJz21Y8umG', 'data': b'x\
↪xda\x95\x9a\x0bpT\xd5\x1...
```

RiRi5 inputs

RiRi5 can have the same inputs as RATP.

A scene grid of voxels must have exactly 9 angle classes for its leaf angle distribution, and at least one empty layer above the canopy.

Triangles mesh inputs are processed through the RATP pipeline, then is converted to RiRi5.

[9]:
```
import numpy

l_scene = {"LA": numpy.ones([2, 3, 4, 4]), "distrib": [[0.1, 0.2, 0.1, 0.1, 0.05, 0.05,␣
↪0.1, 0.2, 0.1],
                                                        [0.1, 0.1, 0.1, 0.1, 0.05, 0.1,␣
↪0.2, 0.2, 0.05]]}
for i in range(4):
    for j in range(4):
        l_scene["LA"][0][0][i][j] = 0.
        l_scene["LA"][1][0][i][j] = 0.
```

[10]:
```
geometry = {
    "scenes": [l_scene]
}
lighting = LightVegeManager(lightmodel="riri5", environment=environment)
lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
```

[21]:
```
print(lighting.riri5_intercepted_light)
```

```
[[[[  0.           0.           0.           0.        ]
   [  0.           0.           0.           0.        ]
   [  0.           0.           0.           0.        ]
   [  0.           0.           0.           0.        ]]

  [[208.05400092 208.05400092 208.05400092 208.05400092]
   [208.05400092 208.05400092 208.05400092 208.05400092]
   [208.05400092 208.05400092 208.05400092 208.05400092]
   [208.05400092 208.05400092 208.05400092 208.05400092]]

  [[ 33.75896074  33.75896074  33.75896074  33.75896074]
   [ 33.75896074  33.75896074  33.75896074  33.75896074]
   [ 33.75896074  33.75896074  33.75896074  33.75896074]
   [ 33.75896074  33.75896074  33.75896074  33.75896074]]]


 [[[  0.           0.           0.           0.        ]
   [  0.           0.           0.           0.        ]
   [  0.           0.           0.           0.        ]
   [  0.           0.           0.           0.        ]]

  [[207.34450322 207.34450322 207.34450322 207.34450322]
   [207.34450322 207.34450322 207.34450322 207.34450322]
   [207.34450322 207.34450322 207.34450322 207.34450322]
   [207.34450322 207.34450322 207.34450322 207.34450322]]

  [[ 33.2079505   33.2079505   33.2079505   33.2079505 ]
   [ 33.2079505   33.2079505   33.2079505   33.2079505 ]
   [ 33.2079505   33.2079505   33.2079505   33.2079505 ]
   [ 33.2079505   33.2079505   33.2079505   33.2079505 ]]]]
```

[22]:
```
print(lighting.riri5_transmitted_light)
```

```
[[[500.         500.         500.         500.        ]
  [500.         500.         500.         500.        ]
  [500.         500.         500.         500.        ]
  [500.         500.         500.         500.        ]]

 [[ 84.60149586  84.60149586  84.60149586  84.60149586]
  [ 84.60149586  84.60149586  84.60149586  84.60149586]
  [ 84.60149586  84.60149586  84.60149586  84.60149586]
  [ 84.60149586  84.60149586  84.60149586  84.60149586]]

 [[ 17.63458462  17.63458462  17.63458462  17.63458462]
```

(continues on next page)

(continued from previous page)

```
 [ 17.63458462  17.63458462  17.63458462  17.63458462]
 [ 17.63458462  17.63458462  17.63458462  17.63458462]
 [ 17.63458462  17.63458462  17.63458462  17.63458462]]]
```

[12]:
```
geometry = {
    "scenes": [organized_triangles]
}

lighting.build(geometry=geometry)
lighting.run(energy=energy, hour=hour, day=day)
```

[13]:
```
print(lighting.riri5_intercepted_light)
```

```
[[[[0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   ...
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]]

  [[0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   ...
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]]

  [[0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   ...
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]]

  ...

  [[0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   ...
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]]

  [[0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   ...
   [0. 0. 0. ... 0. 0. 0.]
```

(continues on next page)

(continued from previous page)

```
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]]

  [[0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   ...
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]
   [0. 0. 0. ... 0. 0. 0.]]]]
```

[14]:
```
print(lighting.riri5_transmitted_light)
```

```
[[[500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  ...
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]]

 [[500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  ...
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]]

 [[500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  ...
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]]

 ...

 [[500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  ...
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]]

 [[500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  [500. 500. 500. ... 500. 500. 500.]
  ...
  [500. 500. 500. ... 500. 500. 500.]
```

(continues on next page)

(continued from previous page)

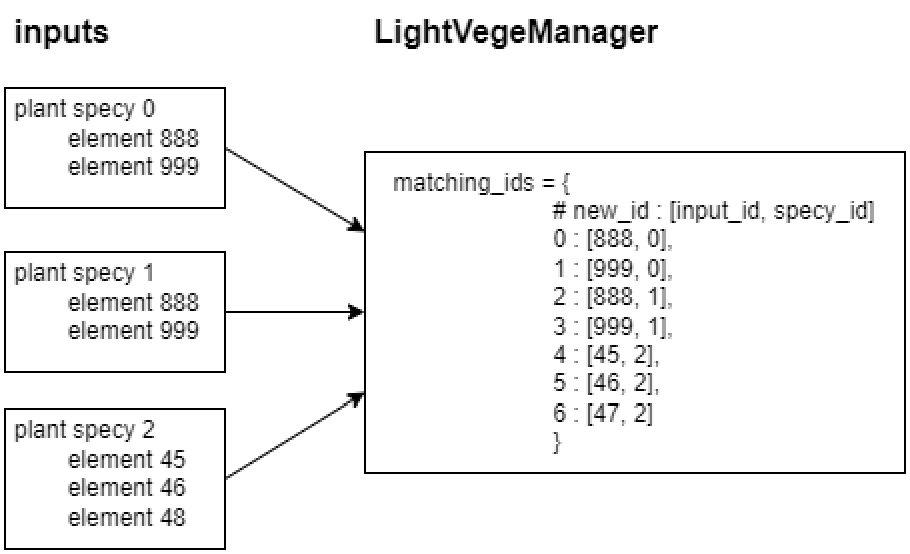
```
 [500. 500. 500. ... 500. 500. 500.]
 [500. 500. 500. ... 500. 500. 500.]]

[[500. 500. 500. ... 500. 500. 500.]
 [500. 500. 500. ... 500. 500. 500.]
 [500. 500. 500. ... 500. 500. 500.]
 ...
 [500. 500. 500. ... 500. 500. 500.]
 [500. 500. 500. ... 500. 500. 500.]
 [500. 500. 500. ... 500. 500. 500.]]]
```

Organizing the inputs

They are two levels of possible organization: - species - organs

Each triangle are bound to a specy ID and a organ ID. Each specy is represented as a input scene. The organs ID are set inside each scene depending on its format.



An example with several scenes with the same organs ID and CARIBU.

[22]:
```
scene1 = {
    111: [random_triangle_generator(spheresize=spheresize) for i in range(20)],
    222: [random_triangle_generator(spheresize=spheresize) for i in range(30)],
    333: [random_triangle_generator(spheresize=spheresize) for i in range(10)],
}

scene2 = {
    111: [random_triangle_generator(spheresize=spheresize) for i in range(20)],
    222: [random_triangle_generator(spheresize=spheresize) for i in range(30)],
    333: [random_triangle_generator(spheresize=spheresize) for i in range(10)],
}

scene3 = {
```

(continues on next page)

(continued from previous page)

```
    111: [random_triangle_generator(spheresize=spheresize) for i in range(20)],
    222: [random_triangle_generator(spheresize=spheresize) for i in range(30)],
    333: [random_triangle_generator(spheresize=spheresize) for i in range(10)],
}
```

We have 3 species

[23]:
```
scenes = [scene1, scene2, scene3]
geometry = { "scenes": scenes }
```

[24]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=geometry)

# compute the lighting
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
```

```
   Day  Hour  Organ  VegetationType         Area    par Eabs      par Ei
0  264    15    111               0   865.848719  367.620782  432.495038
1  264    15    222               0   904.093391  343.903742  404.592638
2  264    15    333               0   500.476649  374.891932  441.049332
3  264    15    111               1   719.501340  330.624848  388.970410
4  264    15    222               1  1023.379192  351.641676  413.696090
5  264    15    333               1   129.105369  391.022326  460.026266
6  264    15    111               2  1047.414777  335.930702  395.212591
7  264    15    222               2  1174.044647  349.124635  410.734865
8  264    15    333               2   521.903258  361.052748  424.767939
```

[25]:
```
SceneWidget(lighting.plantGL_light(),
            position=(-50.0, -50.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[25]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'sNiVT3L9ABichdb2gAyBsVr2f', 'data': b'x\
↪xda\x85}\x07|\x8d\xd7\xf...
```

The two level organization are also kept with RATP

[27]:
```
lighting = LightVegeManager(lightmodel="ratp")
lighting.build(geometry=geometry)

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
```

```
     Day  Hour  Organ  VegetationType         Area        PARa  Intercepted  \
```

(continues on next page)

(continued from previous page)

```
0   264.0  15.0    111                1   865.848719  444.431609   438.050846
1   264.0  15.0    222                1   904.093391  439.138654   508.370824
2   264.0  15.0    333                1   500.476649  438.358519   466.758357
3   264.0  15.0    111                2   719.501340  441.984838   374.664274
4   264.0  15.0    222                2  1023.379192  436.324817   446.932608
5   264.0  15.0    333                2   129.105369  442.768841   373.148804
6   264.0  15.0    111                3  1047.414777  437.256257   728.403508
7   264.0  15.0    222                3  1174.044647  443.465415   536.007093
8   264.0  15.0    333                3   521.903258  432.082470   816.087594

   Transmitted   SunlitPAR  SunlitArea   ShadedPAR  ShadedArea
0   438.050846  450.722334  463.402916  356.222905   49.407779
1   508.370824  446.126727  538.617195  351.627301   58.372064
2   466.758357  446.387940  492.094716  351.888516   56.801907
3   374.664274  449.154416  395.429864  352.554837   42.809579
4   446.932608  444.042247  473.516975  347.442668   51.797538
5   373.148804  449.336097  395.819748  352.736518   45.454794
6   728.403508  444.694193  770.345275  349.806482   81.342446
7   536.007093  450.321284  565.173551  355.433568   58.993793
8   816.087594  440.468993  864.084337  345.581276   97.832614
```

```
c:\users\mwoussen\cdd\codes\dev\lightvegemanager\src\lightvegemanager\outputs.py:255:
↪UserWarning: You are merging on int and float columns where the float values are not
↪equal to their int representation.
  trianglesoutputs = pandas.merge(dftriangles, voxels_outputs)
```

Stems

If there are stems elements in the inputs, you can precise their ID and the tool will manage them, depending on the lightmodel: - with CARIBU, the optical parameters associated to the organ won't have a transmission value (rays won't cross the triangle) - with RATP, stems are separated in a new specy with its own leaf angle distribution and their leaf area is divided by 2

We reuse the wheat geometry given by adelwheat

[9]:
```
geometry = {
    "scenes": [g]
}
```

stems are stored in list where each element is a 2-tuple `(organ ID, specy ID)`

[10]:
```
stems = [(19, 0), (34, 0)]
geometry.update({"stems id": stems})
```

[11]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=geometry)
SceneWidget(lighting.plantGL_nolight(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 0.1,
            axes_helper=True)
```

[11]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'jZ9Zexz60T6u6TydCb9uVlZBe', 'data': b'x\
↪xda\x95\x9a\r\x8cT\xd5\x...
```

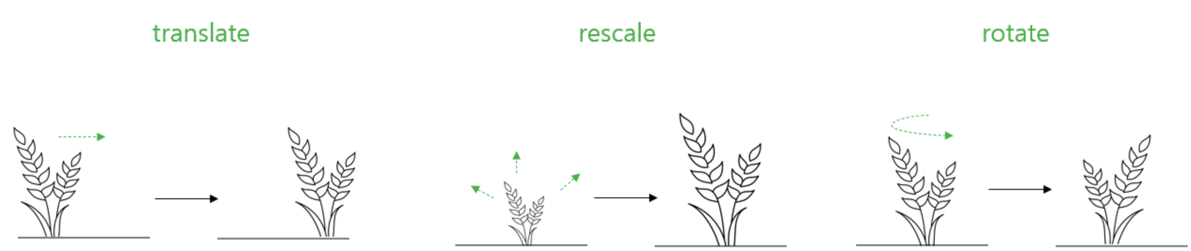
[32]:
```
lighting.run(energy=energy, hour=hour, day=day)
print(lighting.elements_outputs)
SceneWidget(lighting.plantGL_light(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 0.1,
            axes_helper=True)
```

	Day	Hour	Organ	VegetationType	Area	par Eabs	par Ei
0	264	15	19	0	0.000228	194.070089	215.633432
1	264	15	34	0	0.000013	93.806573	104.229526
2	264	15	813	0	0.000194	458.967637	539.961926
3	264	15	814	0	0.000240	392.729426	462.034618
4	264	15	51	0	0.000284	350.644177	412.522561

[32]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'X6MEhdoPOJgZ1bvsYLGzPEHsk', 'data': b'x\
↪xda\x95\x9a{p\x15\xd5\x1...
```

Geometric transformations

You can apply geometric transformations on some of the inputs scenes. We have currently 3 available transformations - translation by a vector - rescale by a factor or following scenes metric unit - rotation on the xy plane



Transformations are stored in a dict, which is stored at key `"transformations"` in the geometry entry. Structure of transformations :

```
transformations = {
    "scenes unit": { specy ID: "metric unit", ...},
    "rescale": { specy ID: float, ...},
    "translate": { specy ID: 3-tuple (x,y,z), ...},
    "xyz orientation":  { specy ID: "x+ = NEWS", ...},
}
```

Main scenes

[13]:
```
spheresize = (2., 1.)
scene1 = {
    0: [random_triangle_generator(spheresize=spheresize, worldsize=(0,10)) for i in␣
↪range(20)]
}
scene2 = {
    0: [random_triangle_generator(spheresize=spheresize, worldsize=(10,20)) for i in␣
↪range(20)]
}

geometry = {"scenes": [scene1, scene2] }
```

[14]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=geometry)
SceneWidget(lighting.plantGL_nolight(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 50.,
            axes_helper=True)
```

[14]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'DOLcNoqMANQw6VQcJjZQN4EVH', 'data': b'x\
↪xda\x85Y{TUu\x16\xbe\x80...
```

Translate

The translation vector is a 3-tuple (x,y,z). Transformations is a dict in the geometry dict.

[15]:
```
tvec = (10., -10., 10.)
transformations = {
    "translate": {
        0: tvec
    }
}
geometry = {"scenes": [scene1, scene2] , "transformations": transformations}
```

[16]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=geometry)
SceneWidget(lighting.plantGL_nolight(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 50.,
            axes_helper=True)
```

[16]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'WtsDD3lUn5IdvzQMKP8sVF7J8', 'data': b'x\
↪xda\x8d\x99\rTTe\x1a\xc7...
```

Rescale following metric unit

You can precise the metric unit of each scene from this list: "mm", "cm", "dm", "m", "dam", "hm", "km". By default the merged scene is in m but you can change its unit when you create an instance.

[17]:
```
transformations = {
    "scenes unit": {
        0: "dm"
    }
}
geometry = {"scenes": [scene1, scene2] , "transformations": transformations}
```

[18]:
```
lighting = LightVegeManager(lightmodel="caribu", main_unit="m")
lighting.build(geometry=geometry)
SceneWidget(lighting.plantGL_nolight(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 50.,
            axes_helper=True)
```

[18]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'xZEoaWgcFvXiGSLzBxMdGtVPm', 'data': b'x\
↪xda\x85\x98\t\\\x95e\x16...
```

Rescale by a scalar factor

[19]:
```
transformations = {
    "rescale": {
        0: 2.,
    }
}
geometry = {"scenes": [scene1, scene2] , "transformations": transformations}
```

[20]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=geometry)
SceneWidget(lighting.plantGL_nolight(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 50.,
            axes_helper=True)
```

[20]:
```
SceneWidget(axes_helper=True, scenes=[{'id': '5z82zPETDhTAdHCoB1NtdJC6R', 'data': b'x\
↪xda\x8d\x99{T\x94\xe5\x1...
```

Rotate

Finally, you can also rotate the scene around the z axis, in order to match the x+ convention for each input scene. You have the choice between: - `"x+ = N"` - `"x+ = S"` - `"x+ = E"` - `"x+ = W"`

The merged scene convention is x+ = N, which the convention in RATP and CARIBU.

[21]:
```
transformations = {
    "xyz orientation": {
        0: "x+ = S",
        1: "x+ = E",
    }
}
geometry = {"scenes": [scene1, scene2] , "transformations": transformations}
```

[22]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=geometry)
SceneWidget(lighting.plantGL_nolight(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 50.,
            axes_helper=True)
```

[22]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'vEFGxUQjZl3lco2qrN9j8zsDG', 'data': b'x\
↪xda\x8d\x99\x0bTUe\x16\x...
```

2.6.4 Light models options: how to set up the light models

Content

- CARIBU
  - Computing the sun position
  - Grid of virtual sensors
  - Other parameters
- RATP
  - Leaf angle distribution
  - Triangles tesselation in a grid
  - Other parameters

Introduction

During our use of lightvegemanager, we added special features for each known light models. This notebook gives you a small introduction to them.

The parameters of those features are stored in a dict.

[1]:
```
import os
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
from lightvegemanager.trianglesmesh import random_triangle_generator
```

CARIBU

This is the complete parameters you can provide with CARIBU:

```
caribu_args = {
                    "sun algo" : "ratp",
                    "sun algo" : "caribu",

                    "caribu opt" : {
                                    band0 = (reflectance, transmittance),
                                    band1 = (reflectance, transmittance),
                                    ...
                                    },
                    "debug" : bool,
                    "soil mesh" : bool,
                    "sensors" : ["grid", dxyz, nxyz, orig, vtkpath, "vtk"]
                }
```

Computing the sun position

In order to compute the sun position, you can use either the algorithm from RATP or CARIBU. The (x, y, z) output is formatted in CARIBU format.

[2]:
```
caribu_args = { "sun algo" : "caribu" }

lighting = LightVegeManager(lightmodel="caribu", lightmodel_parameters=caribu_args)
lighting.build(geometry=[(0., 0., 0.), (1., 0., 0.), (1., 1., 1.)])
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)

sun_caribu = lighting.sun
print(sun_caribu)
```

```
(0.33506553253259913, -0.8798617206271511, -0.3370080733212115)
```

[3]:
```
caribu_args = { "sun algo" : "ratp" }

lighting = LightVegeManager(lightmodel="caribu", lightmodel_parameters=caribu_args)
```

(continues on next page)

(continued from previous page)

```
lighting.build(geometry=[(0., 0., 0.), (1., 0., 0.), (1., 1., 1.)])
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)

sun_ratp = lighting.sun
print(sun_ratp)
```

```
(0.33241183146897624, -0.8800565622452903, -0.3391206592769639)
```

[4]:
```
dist = (sum([ (x-y)**2 for x,y in zip(sun_ratp, sun_caribu) ])) ** (1/2)
print("euclidean dist = ",dist," m")
```

```
euclidean dist =  0.003397515564596359  m
```

Grid of virtual sensors

If you can to match a grid of voxels, you can generate a set of virtual sensors following a 3D grid. You need to precise the dimension of the grid: - dxyz: `[dx, dy, dz]` size of one voxel - nxyz: `[nx, ny, nz]` number of voxels on each xyz axis - orig: `[0x, 0y, 0z]` origin point of the grid

Optionnaly, you can write a geometric visualisation of the sensors in VTK format. You need to provide the file path and the flag `"vtk"`.

[5]:
```
# grid dimensions
dxyz = [1.] * 3
nxyz = [5, 5, 7]
orig = [0.] * 3
```

[6]:
```
# random triangles
nb_triangles = 50
spheresize = (1., 0.3) # vertices of triangles are the sphere surface
triangles = []
for i in range(nb_triangles):
    triangles.append(random_triangle_generator(worldsize=(0., 5.),␣
↪spheresize=spheresize))
```

[7]:
```
caribu_args = { "sensors" : ["grid", dxyz, nxyz, orig] }

lighting = LightVegeManager(lightmodel="caribu", lightmodel_parameters=caribu_args,␣
↪environment={"infinite":True})
lighting.build(geometry=triangles)
```

You can visualize the grid of sensors with plantGL through the method `plantGL_sensors`

[8]:
```
SceneWidget(lighting.plantGL_sensors(),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[8]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'PeUbme32J7raIm5wbjhizJGRd', 'data': b'x\
↪xda\x85\x9ai\xbf\x15G\x1...
```

The lighting results are stored in `sensors_outputs`

[9]:
```python
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)

print(lighting.sensors_outputs)
```


```
{'par': {0: 0.5357183881257019, 1: 0.4686494843618967, 2: 0.43585410695767945, 3: 0.
↪7773054342426935, 4: 0.7653565097977126, 5: 0.8234079750501035, 6: 0.4758490954881396,␣
↪7: 0.5623902514683649, 8: 0.6648303684704121, 9: 0.7523744755517257, 10: 0.
↪757628513619047, 11: 0.7759095328197766, 12: 0.5042164344666725, 13: 0.
↪5273952940619577, 14: 0.618460535502636, 15: 0.7471856369784272, 16: 0.
↪7002461334186968, 17: 0.8083881379807872, 18: 0.5731391756180516, 19: 0.
↪6005392197593685, 20: 0.5830230691495272, 21: 0.668598834740319, 22: 0.
↪7491856052337844, 23: 0.7778896638896193, 24: 0.45941263771422963, 25: 0.
↪5231994757990336, 26: 0.5730702589058677, 27: 0.6802830730422291, 28: 0.
↪768563206195468, 29: 0.8783160299524551, 30: 0.4879256729568832, 31: 0.
↪3986838108537597, 32: 0.5107879595674222, 33: 0.7104544959233176, 34: 0.
↪6104289235914225, 35: 0.8335714361222757, 36: 0.49402264002060914, 37: 0.
↪5236661627298473, 38: 0.5559437694123442, 39: 0.7801809144392181, 40: 0.
↪7482355370820352, 41: 0.8703959530204384, 42: 0.4437212327094782, 43: 0.
↪48943914843165776, 44: 0.6265471398912993, 45: 0.6242676962673399, 46: 0.
↪6242541832847409, 47: 0.8029465687456536, 48: 0.4552489809791363, 49: 0.
↪5706269863386113, 50: 0.6707753329147661, 51: 0.6173647331272717, 52: 0.
↪5817093348224309, 53: 0.8761273435958801, 54: 0.39061205289281303, 55: 0.
↪4876144245567773, 56: 0.6081462607497906, 57: 0.6490748578167109, 58: 0.
↪6112989310198034, 59: 0.9081399625253275, 60: 0.4382387194995509, 61: 0.
↪24511969967129915, 62: 0.5564013821314899, 63: 0.6811375757235048, 64: 0.
↪770734538810677, 65: 0.6876474556435463, 66: 0.4333876306307797, 67: 0.
↪5117511864924423, 68: 0.5452643050348251, 69: 0.5855297120040052, 70: 0.
↪7679378169458118, 71: 0.7374054106105965, 72: 0.46718762617607673, 73: 0.
↪515406942360565, 74: 0.5498630958709402, 75: 0.38157016078052153, 76: 0.
↪6230183331063252, 77: 0.9297880197486428, 78: 0.42012696485414797, 79: 0.
↪43441190921900263, 80: 0.576547648435128, 81: 0.575563926783583, 82: 0.
↪5587431661289568, 83: 0.9428844615982549, 84: 0.40797647513225416, 85: 0.
↪32045180842520204, 86: 0.3860501936085704, 87: 0.6035746806015782, 88: 0.
↪6761415094988514, 89: 0.9147231881446279, 90: 0.35723846481058197, 91: 0.
↪29971902254716964, 92: 0.4617676006672053, 93: 0.6670186733643282, 94: 0.
↪7729889949287282, 95: 0.8223626142139061, 96: 0.43689040185742767, 97: 0.
↪5034266289477778, 98: 0.614485119215983, 99: 0.6753609146208617, 100: 0.
↪8209389374222003, 101: 0.88137358368075, 102: 0.5060744695288328, 103: 0.
↪5006922321596806, 104: 0.5450143532935812, 105: 0.669825913948119, 106: 0.
↪7948585486270389, 107: 0.9497236962858565, 108: 0.5240235293870577, 109: 0.
↪5291828813059647, 110: 0.5584025594129878, 111: 0.7009911880575855, 112: 0.
↪6644575143621863, 113: 0.896722396622886, 114: 0.48066310311501675, 115: 0.
↪45339172597584754, 116: 0.4928664818657807, 117: 0.6921498380364002, 118: 0.
↪6582322232034805, 119: 0.8062622917317241, 120: 0.4165301298570477, 121: 0.
↪47752211531730254, 122: 0.5422072915168136, 123: 0.7320985553256096, 124: 0.
```


(continues on next page)

(continued from previous page)

```
↪760961505134305, 125: 0.8425522336315453, 126: 0.48285036844495494, 127: 0.
↪4897295356464291, 128: 0.6751897497018744, 129: 0.7650357314236249, 130: 0.
↪6888417962718921, 131: 0.9099594684378505, 132: 0.4801370709108044, 133: 0.
↪5233846696457297, 134: 0.5942217135899492, 135: 0.7384234333722701, 136: 0.
↪8064837117567087, 137: 0.955340358423788, 138: 0.5755258055313021, 139: 0.
↪5351956047253674, 140: 0.5430459909905389, 141: 0.7603523762894485, 142: 0.
↪8096038729562012, 143: 0.9210479253387107, 144: 0.49256349385820036, 145: 0.
↪4352151194578986, 146: 0.4353486297408457, 147: 0.7458275318718196, 148: 0.
↪7991276473463365, 149: 0.9117731862505419}}
```

You can also visualize the results in the plantGL scene

[10]:
```
SceneWidget(lighting.plantGL_sensors(light=True),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[10]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'h4EUPTlkwcC9SjqSa9KdhgS4D', 'data': b'x\
↪xda\x85\x9bk\x7fVG\x15\x...
```

[11]:
```
SceneWidget(lighting.plantGL_sensors(light=True) + lighting.plantGL_nolight(),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[11]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'whSP4mA3wlgKFoL4Li9pam9AQ', 'data': b'x\
↪xda\x8d\x9cy\x98\x15\xc5...
```

Other parameters

In additional features, you can activate the debug mode in CARIBU, which describe the internal steps.

[6]:
```
caribu_args = { "debug" : True }

lighting = LightVegeManager(lightmodel="caribu", lightmodel_parameters=caribu_args)
lighting.build(geometry=[(0., 0., 0.), (1., 0., 0.), (1., 1., 1.)])

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

```
Prepare scene 1
done
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[6], line 9
      7 hour = 15
```

(continues on next page)

(continued from previous page)

```
      8 day = 264
----> 9 lighting.run(energy=energy, hour=hour, day=day)

File c:\users\mwoussen\cdd\codes\dev\lightvegemanager\src\lightvegemanager\tool.py:533,␣
↪in LightVegeManager.run(self, energy, day, hour, parunit, truesolartime, id_sensors)
    531 if sun_sky_option == "mix":
    532     start = time.time()
--> 533     raw_sun, aggregated_sun = run_caribu(*arg)
    534     arg[0] = c_scene_sky
    535     raw_sky, aggregated_sky = run_caribu(*arg)

File c:\users\mwoussen\cdd\codes\dev\lightvegemanager\src\lightvegemanager\CARIBUinputs.
↪py:370, in run_caribu(c_scene, direct_active, infinite, sensors, energy)
    329 """runs caribu depending on input options
    330
    331 :param c_scene: instance of CaribuScene containing geometry, light source(s),␣
↪opt etc...
   (...)
    367 :rtype: dict of dict, dict of dict
    368 """
    369 if sensors is None :
--> 370     raw, aggregated = c_scene.run(direct=direct_active, infinite=infinite)
    371 else :
    372     raw, aggregated = c_scene.run(direct=direct_active, infinite=infinite,
    373                                                         sensors=sensors)

File ~\AppData\Local\miniconda3\envs\mobidivpy37\lib\site-packages\alinea.caribu-8.0.7-
↪py3.8.egg\alinea\caribu\CaribuScene.py:568, in CaribuScene.run(self, direct, infinite,␣
↪d_sphere, layers, height, screen_size, screen_resolution, sensors, split_face,␣
↪simplify)
    566         self.canfile = os.path.join(self.tempdir,'cscene.can')
    567         self.optfile = os.path.join(self.tempdir,'band0.opt')
--> 568         ␣
↪write_scene(triangles, materials, canfile = self.canfile, optfile = self.optfile)
    570 else:
    571     # self.materialvalues is a cache for the computation of the material list
    572     materials = self.materialvalues

File ~\AppData\Local\miniconda3\envs\mobidivpy37\lib\site-packages\alinea.caribu-8.0.7-
↪py3.8.egg\alinea\caribu\caribu.py:177, in write_scene(triangles, materials, canfile,␣
↪optfile)
    175 o_string, labels = opt_string_and_labels(materials)
    176 can_string = triangles_string(triangles, labels)
--> 177 open(canfile,'w').write(can_string)
    178 open(optfile,'w').write(o_string)

FileNotFoundError: [Errno 2] No such file or directory: './caribuscene_2672400716080\\
↪cscene.can'
```

You can also use the soilmesh option and get the lighting hitting the soil. The method `soilenergy` get you access to its result.

[12]:
```
caribu_args = { "soil mesh" : True }

lighting = LightVegeManager(lightmodel="caribu", lightmodel_parameters=caribu_args)
lighting.build(geometry=[(0., 0., 0.), (1., 0., 0.), (1., 1., 1.)])

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)

print(lighting.soilenergy)
```

```
{'Qi': 0.6750540873627096, 'Einc': 0.6750540873627096}
```

RATP

This is the complete parameters you can provide with CARIBU:

```
ratp_args = {
                # Grid specifications
                "voxel size" : [dx, dy, dz],
                "voxel size" : "dynamic",

                "full grid" : bool,

                "origin" : [xorigin, yorigin, zorigin],
                "origin" : [xorigin, yorigin],

                "number voxels" : [nx, ny, nz],
                "grid slicing" : "ground = 0."
                "tesselation level" : int

                # Leaf angle distribution
                "angle distrib algo" : "compute global",
                "angle distrib algo" : "compute voxel",
                "angle distrib algo" : "file",

                "nb angle classes" : int,
                "angle distrib file" : filepath,

                # Vegetation type
                "soil reflectance" : [reflectance_band0, reflectance_band1, ...],
                "reflectance coefficients" : [reflectance_band0, reflectance_band1, ...],
                "mu" : [mu_scene0, mu_scene1, ...]
            }
```

Leaf angle distribution

Leaf angle distribution can be generated in 3 ways: - from a file, one distribution per specy - global and dynamically, it generates a distribution from a triangles mesh for each specy - per voxel and dynamically, it generates a distribution from the triangles located in each voxel

[2]:
```
# random triangles
nb_triangles = 50
spheresize = (1., 0.3) # vertices of triangles are the sphere surface
worldsize = (0., 5.)
triangles = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for i␣
↪in range(nb_triangles)]
```

File

You need the flag "`file`" and to specify the file path.

[15]:
```
filepath = os.path.join(os.path.dirname(os.path.abspath("")), "data", "luzerne_angle_
↪distrib.data")
ratp_parameters = { "angle distrib algo" : "file", "angle distrib file" : filepath }

# initialize the instance
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)

# build the scene
lighting.build(geometry=triangles)

print(lighting.leafangledistribution)
```

```
{'global': [[0.1382, 0.1664, 0.1972, 0.1925, 0.1507, 0.0903, 0.0425, 0.0172, 0.005]]}
```

Global distribution

You need the flag "`compute global`" and to specify the number of angle classes you need.

[3]:
```
ratp_parameters = { "angle distrib algo" : "compute global", "nb angle classes" : 9 }

# initialize the instance
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)

# build the scene
lighting.build(geometry=triangles)

print(lighting.leafangledistribution)
```

```
{'global': [[0.0, 0.01403472520994376, 0.14553389876181141, 0.13870880399511626, 0.
↪17786938035077418, 0.06842434118299212, 0.09608726894836216, 0.20983153542452182, 0.
↪14951004612647875]]}
```

Local distribution

You need the flag "compute voxel" and to specify the number of angle classes you need. You will get one distribution for each voxel of your grid and each specy.

[6]:
```
ratp_parameters = {
                    "voxel size" : [1., 1., 1.],
                    "angle distrib algo" : "compute voxel",
                    "nb angle classes" : 9
                    }

# initialize the instance
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)

# build the scene
lighting.build(geometry=triangles)

print("Global")
print(lighting.leafangledistribution["global"])
print("\n\n Local")
for a in lighting.leafangledistribution["voxel"]:
    print(a[0])
```

```
Global
[[0.009674637223767685, 0.06468549489243054, 0.10705660994810652, 0.10196105003699654, 0.
↪07575759861707439, 0.3044721953841436, 0.06446975879644407, 0.14828246866341796, 0.
↪12364018643761852]]


 Local
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0.         0.         0.55805526 0.44194474 0.         0.
 0.         0.         0.        ]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0.         0.         0.41000514 0.58999486 0.         0.
 0.         0.         0.        ]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

(continues on next page)

(continued from previous page)

```
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0.         0.         0.58893012 0.41106988 0.         0.
 0.         0.         0.        ]
[0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0.         0.30170822 0.         0.         0.         0.
 0.         0.         0.69829178]
[0.         0.         0.         0.17828753 0.         0.
 0.         0.         0.82171247]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 1. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1.]
[0. 0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

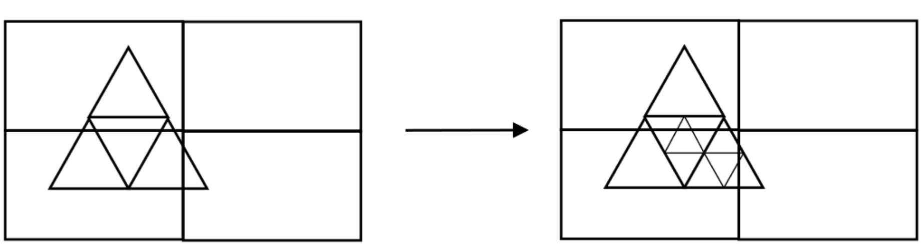
For visualization of the situation

[7]:
```
SceneWidget(lighting.plantGL_nolight(printtriangles=True, printvoxels=True),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[7]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'Rp6rgkWOhuO88XZzffdSscV0V', 'data': b'x\
↪xda\x8d\x9b{\x9c\x8d\xd5...
```

Triangles tesselation in a grid

You can reduce the error while transferring a triangle mesh to a voxel mesh by subdividing triangles across multiple voxels.



You only need to precise how many times you want to subdivide the triangles.

[21]:
```
ratp_parameters = { "voxel size" : [1., 1., 1.], "tesselation level" : 7 }

# initialize the instance
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)

# build the scene
lighting.build(geometry=triangles)

SceneWidget(lighting.plantGL_nolight(printtriangles=True, printvoxels=True),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[21]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'w9mu1jQPBqZrSzDk4B45B9Wav', 'data': b'x\
↪xda\x94}\x07\x9c^E\xd5~\...
```

Other parameters

By default, the number of voxels is dynamically computed following the voxel size and mesh limits, but you can force its number.

Voxel size can also be dynamically computed and is based on 3 times the longest triangle.

2.6.5 Output formats and transfer methods

Content

- Main light outputs
- transfer results to l-egume
- transfer results to CN-Wheat

Introduction

This notebook will present the different output formats given by the tool.

[1]:
```
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
from lightvegemanager.trianglesmesh import random_triangle_generator
```

Main light outputs: Pandas dataframe

Outputs are stored in at least two different scales, by triangle or voxel, and by organ. We will use a set of random triangles as an illustration.

[2]:
```
# random triangles
nb_triangles = 20
spheresize = (1., 0.3) # vertices of triangles are the sphere surface
worldsize = (0., 5.)
triangles = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for i␣
↪in range(nb_triangles)]
```

We compute one iteration with CARIBU

[3]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=triangles)

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

Results for each triangle

[4]:
```
print(type(lighting.triangles_outputs),"\n")
print(lighting.triangles_outputs)
```

```
<class 'pandas.core.frame.DataFrame'>

    Day  Hour  Triangle  Organ  VegetationType      Area    par Eabs  \
0   264    15         0      0               0  0.536966  432.528062
1   264    15         1      0               0  0.504415  473.973825
2   264    15         2      0               0  0.314083  466.696396
3   264    15         3      0               0  0.057297  427.303344
4   264    15         4      0               0  0.266138  346.795923
5   264    15         5      0               0  0.318387  359.266951
6   264    15         6      0               0  0.089070  480.230969
7   264    15         7      0               0  0.436727  383.952401
8   264    15         8      0               0  0.094307  380.840868
9   264    15         9      0               0  0.501522  388.825138
10  264    15        10      0               0  0.051670  382.490896
11  264    15        11      0               0  0.193144  407.201836
12  264    15        12      0               0  0.585857  414.603624
13  264    15        13      0               0  0.139755  427.431809
14  264    15        14      0               0  0.805939  317.704624
15  264    15        15      0               0  0.104366  420.622125
16  264    15        16      0               0  0.214229  461.347939
17  264    15        17      0               0  0.273954  426.103905
18  264    15        18      0               0  0.986628  376.478502
19  264    15        19      0               0  0.303850  469.840116

        par Ei
0   508.856544
1   557.616265
```

(continues on next page)

(continued from previous page)

```
2    549.054584
3    502.709816
4    407.995204
5    422.667001
6    564.977611
7    451.708707
8    448.048080
9    457.441339
10   449.989290
11   479.060984
12   487.768969
13   502.860951
14   373.770146
15   494.849559
16   542.762281
17   501.298712
18   442.915885
19   552.753077
```

We can try to group multiple sets of triangles

[5]:
```
nb_triangles = 10
triangles1 = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for␣
↪i in range(nb_triangles)]

nb_triangles = 9
triangles2 = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for␣
↪i in range(nb_triangles)]

nb_triangles = 8
triangles3 = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for␣
↪i in range(nb_triangles)]
```

[6]:
```
scene = {0: triangles1, 1: triangles2, 2: triangles3}

lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry={"scenes" : [scene] })

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[7]:
```
print(lighting.triangles_outputs)
```

```
    Day  Hour  Triangle  Organ  VegetationType      Area    par Eabs  \
0   264    15         0      0               0  0.347893  466.934112
1   264    15         1      0               0  0.230175  343.491693
2   264    15         2      0               0  0.236752  406.441366
3   264    15         3      0               0  0.055911  431.671508
4   264    15         4      0               0  2.047361  349.189980
5   264    15         5      0               0  0.132022  305.537738
```

(continues on next page)

(continued from previous page)

```
6   264    15         6      0                 0  0.179736  347.226774
7   264    15         7      0                 0  0.353962  391.605479
8   264    15         8      0                 0  1.065609  441.957469
9   264    15         9      0                 0  0.196493  334.393347
10  264    15        10      1                 0  0.014315  335.982872
11  264    15        11      1                 0  0.047547  409.832578
12  264    15        12      1                 0  0.932423  417.792230
13  264    15        13      1                 0  0.078560  330.173348
14  264    15        14      1                 0  0.037776  413.308224
15  264    15        15      1                 0  1.845328  393.111328
16  264    15        16      1                 0  0.212917  402.681041
17  264    15        17      1                 0  0.367550  360.846221
18  264    15        18      1                 0  0.104118  339.165224
19  264    15        19      2                 0  0.051821  481.839430
20  264    15        20      2                 0  0.172513  316.689490
21  264    15        21      2                 0  0.274700  360.822712
22  264    15        22      2                 0  0.150917  274.216253
23  264    15        23      2                 0  0.028843  372.272165
24  264    15        24      2                 0  0.643578  373.151609
25  264    15        25      2                 0  0.094779  340.286767
26  264    15        26      2                 0  0.069499  388.224866

        par Ei
0   549.334249
1   404.107874
2   478.166313
3   507.848832
4   410.811741
5   359.456163
6   408.502087
7   460.712328
8   519.949964
9   393.403938
10  395.273967
11  482.155974
12  491.520271
13  388.439233
14  486.244970
15  462.483915
16  473.742402
17  424.524966
18  399.017911
19  566.869918
20  372.575870
21  424.497308
22  322.607357
23  437.967253
24  439.001893
25  400.337373
26  456.735136
```

And the grouped results

[8]:
```
print(lighting.elements_outputs)
```

```
   Day  Hour  Organ  VegetationType      Area    par Eabs      par Ei
0  264    15      0               0  4.845914  382.756535  450.301806
1  264    15      1               0  3.640534  394.037183  463.573157
2  264    15      2               0  1.486650  356.659092  419.598932
```

With RATP, you have another output for each voxel

[9]:
```
scene = {0: triangles1, 1: triangles2, 2: triangles3}

lighting = LightVegeManager(lightmodel="ratp")
lighting.build(geometry={"scenes" : [scene] })

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[10]:
```
print(lighting.triangles_outputs)
```

```
    Triangle  Organ  Voxel  VegetationType  primitive_area    Day  Hour  Nx  \
0          0      0    1.0               1        0.347893  264.0  15.0   1
1          1      0    1.0               1        0.230175  264.0  15.0   1
2          2      0    1.0               1        0.236752  264.0  15.0   1
3          3      0    1.0               1        0.055911  264.0  15.0   1
4          4      0    1.0               1        2.047361  264.0  15.0   1
5          5      0    1.0               1        0.132022  264.0  15.0   1
6          6      0    1.0               1        0.179736  264.0  15.0   1
7          7      0    1.0               1        0.353962  264.0  15.0   1
8          8      0    1.0               1        1.065609  264.0  15.0   1
9          9      0    1.0               1        0.196493  264.0  15.0   1
10        10      1    1.0               1        0.014315  264.0  15.0   1
11        11      1    1.0               1        0.047547  264.0  15.0   1
12        12      1    1.0               1        0.932423  264.0  15.0   1
13        13      1    1.0               1        0.078560  264.0  15.0   1
14        14      1    1.0               1        0.037776  264.0  15.0   1
15        15      1    1.0               1        1.845328  264.0  15.0   1
16        16      1    1.0               1        0.212917  264.0  15.0   1
17        17      1    1.0               1        0.367550  264.0  15.0   1
18        18      1    1.0               1        0.104118  264.0  15.0   1
19        19      2    1.0               1        0.051821  264.0  15.0   1
20        20      2    1.0               1        0.172513  264.0  15.0   1
21        21      2    1.0               1        0.274700  264.0  15.0   1
22        22      2    1.0               1        0.150917  264.0  15.0   1
23        23      2    1.0               1        0.028843  264.0  15.0   1
24        24      2    1.0               1        0.643578  264.0  15.0   1
25        25      2    1.0               1        0.094779  264.0  15.0   1
26        26      2    1.0               1        0.069499  264.0  15.0   1

    Ny  Nz   ShadedPAR   SunlitPAR  ShadedArea  SunlitArea      Area  \
0    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
1    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
2    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
```

(continues on next page)

(continued from previous page)

```
3    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
4    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
5    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
6    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
7    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
8    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
9    1   1  363.182404  461.736481    0.416941    9.556157  9.973098
10   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
11   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
12   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
13   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
14   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
15   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
16   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
17   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
18   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
19   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
20   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
21   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
22   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
23   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
24   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
25   1   1  363.182404  461.736481    0.416941    9.556157  9.973098
26   1   1  363.182404  461.736481    0.416941    9.556157  9.973098

          PARa  Intercepted  Transmitted
0   457.616241     9.127703    44.626312
1   457.616241     9.127703    44.626312
2   457.616241     9.127703    44.626312
3   457.616241     9.127703    44.626312
4   457.616241     9.127703    44.626312
5   457.616241     9.127703    44.626312
6   457.616241     9.127703    44.626312
7   457.616241     9.127703    44.626312
8   457.616241     9.127703    44.626312
9   457.616241     9.127703    44.626312
10  457.616241     9.127703    44.626312
11  457.616241     9.127703    44.626312
12  457.616241     9.127703    44.626312
13  457.616241     9.127703    44.626312
14  457.616241     9.127703    44.626312
15  457.616241     9.127703    44.626312
16  457.616241     9.127703    44.626312
17  457.616241     9.127703    44.626312
18  457.616241     9.127703    44.626312
19  457.616241     9.127703    44.626312
20  457.616241     9.127703    44.626312
21  457.616241     9.127703    44.626312
22  457.616241     9.127703    44.626312
23  457.616241     9.127703    44.626312
24  457.616241     9.127703    44.626312
25  457.616241     9.127703    44.626312
```

(continues on next page)

(continued from previous page)

```
26  457.616241     9.127703    44.626312
```

[11]:
```
print(lighting.voxels_outputs)
```

```
   VegetationType    Day  Hour  Voxel  Nx  Ny  Nz   ShadedPAR   SunlitPAR  \
0             1.0  264.0  15.0    1.0   1   1   1  363.182404  461.736481

   ShadedArea  SunlitArea      Area        PARa  Intercepted  Transmitted
0    0.416941    9.556157  9.973098  457.616241     9.127703    44.626312
```

[12]:
```
print(lighting.elements_outputs)
```

```
     Day  Hour  Organ  VegetationType      Area        PARa  Intercepted  \
0  264.0  15.0      0               1  4.845914  457.616241     9.127703
1  264.0  15.0      1               1  3.640534  457.616241     9.127703
2  264.0  15.0      2               1  1.486650  457.616241     9.127703

   Transmitted   SunlitPAR  SunlitArea   ShadedPAR  ShadedArea
0     9.127703  461.736481    9.556157  363.182404    0.416941
1     9.127703  461.736481    9.556157  363.182404    0.416941
2     9.127703  461.736481    9.556157  363.182404    0.416941
```

l-egume results transfer

There are two possible scenarios: - with RATP, the tool reformats the data types. Grid specifications must match with l-egume internal grid instance - with CARIBU, you need to use virtual sensors following the dimensions of l-egume internal grid.

The id argument is used with you several input scenes but you need to transfer only some of them to your l-egume instance.

In both case, it will return two tables, one with intercepted lighting and another with transmitted lighting in each voxel.

[13]:
```
# grid dimensions
dxyz = [1.] * 3
nxyz = [7, 7, 7]
orig = [-1., -1., 0.]
```

[14]:
```
spheresize = (1., 0.3) # vertices of triangles are the sphere surface
worldsize = (0., 5.)

nb_triangles = 10
triangles1 = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for␣
↪i in range(nb_triangles)]

nb_triangles = 9
triangles2 = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for␣
↪i in range(nb_triangles)]

nb_triangles = 8
triangles3 = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for␣
↪i in range(nb_triangles)]
```

(continues on next page)

(continued from previous page)

```
scene = {0: triangles1, 1: triangles2, 2: triangles3}
```

RATP

Input: l-egume intern grid of leaf area

[15]:
```
ratp_parameters = { "voxel size" : dxyz,
                    "origin" : orig,
                    "number voxels" : nxyz,
                    "full grid" : True}

lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)
lighting.build(geometry={"scenes" : [scene] })

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[16]:
```
SceneWidget(lighting.plantGL_nolight(printtriangles=True, printvoxels=True),
            position=(0., 0., 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[16]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'Eo1pXjKQFbVhh6Ksf54Ry6k6D', 'data': b'x\
↪xda\x8d}\x0b\x94\xadEuf+...
```

[17]:
```
import numpy

m_lais = numpy.zeros([1] + nxyz)

for row in lighting.voxels_outputs.itertuples():
    m_lais[int(row.VegetationType)-1][row.Nz-1][row.Nx-1][row.Ny-1] = row.Area
```

[18]:
```
res_abs_i, res_trans = lighting.to_l_egume(m_lais=m_lais)

print("PARa intercepted")
print(res_abs_i)
print("\n")
print("PARa transmitted")
print(res_trans)
```

```
PARa intercepted
[[[[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
```

(continues on next page)

(continued from previous page)

```
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]]

[[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 4.89933714e-02 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 2.93938100e-01 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 2.57237822e-01
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]]

[[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 3.80286694e-01 8.93628836e-01
  6.55927658e-01 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.29318118e-01 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 5.11650145e-01
  3.74495775e-01 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]]

[[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 5.34315050e-01 1.00000000e-14
  4.15218249e-02 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  4.28144723e-01 5.49222976e-02 1.00000000e-14]
 [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
  1.00000000e-14 1.00000000e-14 2.89329551e-02]
```

(continues on next page)

(continued from previous page)

```
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]]

  [[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.69486284e-01 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 8.16469371e-01 1.00000000e-14
    1.00000000e-14 6.63995743e-02 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]]

  [[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 2.14309216e-01 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 2.51297981e-01 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 6.85566485e-01
    1.00000000e-14 1.11084175e+00 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]]

  [[1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.67224873e-02 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]
   [1.00000000e-14 1.00000000e-14 1.00000000e-14 1.00000000e-14
    1.00000000e-14 1.00000000e-14 1.00000000e-14]]]]


PARa transmitted
[[[1.         1.         1.         1.         1.         1.
   1.        ]
```

(continues on next page)

(continued from previous page)

```
 [1.         1.         1.         1.         1.         1.
  1.        ]
 [1.         1.         1.         1.         1.         1.
  1.        ]
 [1.         1.         1.         1.         1.         1.
  1.        ]
 [1.         1.         1.         1.         1.         1.
  1.        ]
 [1.         1.         1.         1.         1.         1.
  1.        ]
 [1.         1.         1.         1.         1.         1.
  1.        ]]

[[1.         0.99966007 0.99862611 0.99862772 0.99882162 1.
  1.        ]
 [0.99937242 0.99652928 0.99643964 0.99913657 0.99951363 0.99987358
  0.9998582 ]
 [0.99661541 0.9750672  0.97824836 0.99296302 0.99881232 0.99979579
  0.9996115 ]
 [0.9968856  0.98334223 0.88385904 0.97908282 0.99741894 0.99954569
  0.99927342]
 [0.99909747 0.99502355 0.98070908 0.97873741 0.99385464 0.99878031
  1.        ]
 [0.99955082 0.99661982 0.98586112 0.89782482 0.98341918 0.99853724
  0.99973792]
 [1.         0.99890536 0.99653733 0.98721927 0.99558949 0.99994254
  0.99992949]]

[[0.99485987 0.98211366 0.94685709 0.90895796 0.92835289 0.9768011
  0.99335617]
 [0.98958224 0.95989323 0.78321606 0.59019464 0.68427098 0.94649696
  0.98931742]
 [0.99396461 0.97352087 0.89230376 0.90797049 0.93556297 0.97943383
  0.99502289]
 [0.99184281 0.98125339 0.94494903 0.93165594 0.95026821 0.98223406
  0.99335128]
 [0.98969823 0.9849447  0.95014137 0.76321989 0.81169522 0.96606922
  0.98965567]
 [0.99407554 0.99231446 0.976399   0.93636918 0.95371062 0.98325461
  0.99324131]
 [0.99734932 0.99393582 0.9833504  0.97493804 0.9806965  0.99197996
  0.9983626 ]]

[[0.98496985 0.96739489 0.9302929  0.89807343 0.9174937  0.95895547
  0.98020232]
 [0.98125023 0.94733357 0.86565912 0.84225869 0.8774448  0.95282722
  0.98152596]
 [0.97424239 0.93394589 0.72031593 0.88447887 0.90997893 0.9595657
  0.9761579 ]
 [0.97994757 0.96309948 0.90736842 0.8934654  0.89512074 0.94767475
  0.97867441]
 [0.98629206 0.97525859 0.95201224 0.88435513 0.7504518  0.91819239
```

(continues on next page)

(continued from previous page)

```
  0.98110867]
 [0.98893738 0.98108053 0.96626729 0.94061965 0.93254912 0.95972735
  0.97235537]
 [0.98517865 0.97681737 0.96344769 0.95575136 0.95711577 0.96942037
  0.9815892 ]]

[[0.97933716 0.97092551 0.94165611 0.94100547 0.94331664 0.95525873
  0.97785884]
 [0.97096044 0.94943368 0.91685653 0.90953714 0.91383499 0.89041752
  0.96821451]
 [0.9826436  0.95704621 0.91028017 0.92085916 0.93676978 0.95857286
  0.97805393]
 [0.97259653 0.94509262 0.87017393 0.89592469 0.91131461 0.94362009
  0.96967971]
 [0.98114228 0.93692589 0.66466349 0.88666028 0.9184655  0.92707497
  0.97069734]
 [0.97923797 0.97428864 0.9446348  0.95074368 0.95068192 0.96627277
  0.9782179 ]
 [0.98016465 0.97797263 0.95464146 0.94952333 0.94629568 0.97018468
  0.980313  ]]

[[0.95296037 0.95422834 0.94934404 0.93673187 0.93656844 0.93842596
  0.94804531]
 [0.97229284 0.96320361 0.9393397  0.92389566 0.92137164 0.87234128
  0.95597047]
 [0.96675485 0.94391161 0.90375632 0.92214018 0.92659348 0.92107111
  0.96080649]
 [0.95322537 0.92634034 0.80349278 0.84525293 0.90094328 0.86473721
  0.93806767]
 [0.96127474 0.93329871 0.8198998  0.64854169 0.81475776 0.55683774
  0.90463072]
 [0.98215431 0.96413124 0.91450155 0.88721132 0.9184497  0.89321476
  0.96541333]
 [0.98092657 0.96966964 0.9389317  0.95247173 0.94251043 0.95123148
  0.96945715]]

[[0.94756877 0.95514208 0.94929218 0.94613492 0.92572314 0.9404825
  0.93554872]
 [0.97161055 0.95421797 0.92918479 0.93547511 0.92323899 0.93136919
  0.94081557]
 [0.94040775 0.94937837 0.93365628 0.92724371 0.91473806 0.9428041
  0.93166846]
 [0.94083965 0.92754972 0.9003523  0.89309627 0.89577699 0.90264505
  0.92660093]
 [0.94083911 0.92529798 0.88156152 0.84460795 0.84956956 0.85377795
  0.90978062]
 [0.93541592 0.93342227 0.9089222  0.90135753 0.88972384 0.89331508
  0.92612672]
 [0.96249211 0.96975094 0.94652379 0.9468134  0.93272442 0.94823855
  0.95390475]]]
```

CARIBU

Input: l-egume intern grid of leaf area

[19]:
```
caribu_args = { "sensors" : ["grid", dxyz, nxyz, orig] }

lighting = LightVegeManager(lightmodel="caribu", lightmodel_parameters=caribu_args,␣
↪environment={"infinite":True})
lighting.build(geometry={"scenes" : [scene] })

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[20]:
```
SceneWidget(lighting.plantGL_sensors(light=True) + lighting.plantGL_nolight(),
            position=(-0., -0., 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[20]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'wJZ83l0BmoFl6R5sdBeyGBiUB', 'data': b'x\
↪xda\x8d\x9d\r\xbcUE\xb9\...
```

[21]:
```
# plant parameters, those variables are part of l-egume, here is a simplified version␣
↪for our example
list_invar = [{"Hplante": [0.0] * 2}]
list_lstring = [
    {
        0: [0, 0, 0, 0, 0, 0, 0, 0, 0, "dev"],
        1: [0, 0, 0, 0, 0, 0, 0, 0, 0, "sen"],
        2: [1, 0, 0, 0, 0, 0, 0, 0, 0, "dev"],
    }
]
list_dicFeuilBilanR = [
    {"surf": [0.5, 1e-8]},
]

import numpy

m_lais = numpy.zeros([1] + nxyz)
```

[22]:
```
res_trans = lighting.to_l_egume(m_lais=m_lais,
                                list_lstring=list_lstring,
                                list_dicFeuilBilanR=list_dicFeuilBilanR,
                                list_invar=list_invar)
print("PARa transmitted")
print(res_trans)

print("\n")
print("PARa absorbed per plant")
print(list_invar)
```

```
PARa transmitted
[[[1.         1.         1.         1.         1.         1.
   1.        ]
  [1.         1.         1.         1.         1.         1.
   1.        ]
  [1.         1.         1.         1.         1.         1.
   1.        ]
  [1.         1.         1.         1.         1.         1.
   1.        ]
  [1.         1.         1.         1.         1.         1.
   1.        ]
  [1.         1.         1.         1.         1.         1.
   1.        ]
  [1.         1.         1.         1.         1.         1.
   1.        ]]

 [[0.99283465 0.97547655 0.97996241 0.9866237  0.9583532  0.95865713
   0.98694664]
  [0.97774844 0.96530949 0.97812413 0.94335656 0.79359731 0.8288275
   0.97984945]
  [0.95961262 0.96250934 0.97278397 0.91272267 0.7382741  0.79477271
   0.98515375]
  [0.97219276 0.97383604 0.95889506 0.92757533 0.87868432 0.9093107
   0.98973258]
  [0.99499098 0.99379047 0.97594497 0.96094415 0.9626287  0.97785097
   0.9939268 ]
  [0.99696075 0.99603476 0.99520086 0.98950364 0.98841795 0.99438669
   0.99632086]
  [0.99607408 0.99632587 0.99635674 0.99631713 0.99698258 0.99689147
   0.99685284]]

 [[0.97470511 0.94514069 0.93077744 0.93920035 0.9329825  0.92910817
   0.9597604 ]
  [0.93902562 0.84260595 0.83278455 0.88433881 0.86394577 0.88136
   0.94917899]
  [0.92365243 0.80240585 0.77352939 0.81993138 0.7911815  0.84628719
   0.94733873]
  [0.96214209 0.90668456 0.86337504 0.83267538 0.8023391  0.87666842
   0.95844153]
  [0.98205481 0.97935428 0.96394767 0.92029812 0.90199128 0.9400237
   0.97286147]
  [0.98656516 0.98394133 0.96488209 0.9617456  0.97551971 0.97480579
   0.97693207]
  [0.98860163 0.97982773 0.96375513 0.97078053 0.97175977 0.94172195
   0.95478506]]

 [[0.95701018 0.92543249 0.91452955 0.91529581 0.90796187 0.92735326
   0.96004947]
  [0.92750482 0.87879103 0.87139533 0.88794122 0.88021506 0.90711292
   0.94913605]
  [0.91243229 0.85247416 0.84137598 0.83055153 0.82787463 0.8959008
   0.94203007]
  [0.91033121 0.84517546 0.82532049 0.79060523 0.80545506 0.89567693
```

(continues on next page)

(continued from previous page)

```
   0.91514608]
  [0.93257741 0.86079298 0.8449184  0.8523624  0.87150218 0.91578729
   0.9013189 ]
  [0.96779888 0.93102995 0.91071792 0.92115358 0.933688   0.94220112
   0.93189039]
  [0.97735824 0.96935045 0.93242933 0.91134405 0.94099756 0.959761
   0.95889625]]

 [[0.92296813 0.90516463 0.90284731 0.91447234 0.87371474 0.88073977
   0.93560093]
  [0.88270877 0.88940347 0.85396849 0.85397283 0.84917024 0.87384279
   0.91539319]
  [0.87898549 0.8856543  0.83203816 0.82884733 0.86532201 0.89502837
   0.90832493]
  [0.88237416 0.84629472 0.83386876 0.83964468 0.85864715 0.90311898
   0.92454761]
  [0.89126296 0.83400359 0.8388134  0.86334547 0.86850161 0.9055682
   0.93470077]
  [0.92007599 0.8866167  0.88060342 0.90901959 0.91319735 0.91664405
   0.93776939]
  [0.93967084 0.9347306  0.92625581 0.93690253 0.94524498 0.94220458
   0.9435016 ]]

 [[0.85841022 0.74993594 0.76130349 0.85284938 0.86431234 0.87480163
   0.90179267]
  [0.86526185 0.74226062 0.75591914 0.85134443 0.83492443 0.83171408
   0.88010129]
  [0.86923228 0.77258171 0.7676997  0.8318704  0.83804378 0.85347289
   0.87805225]
  [0.87857998 0.816706   0.79000755 0.82842821 0.8698445  0.89948354
   0.90035524]
  [0.89811456 0.86396829 0.85201001 0.87253328 0.90782969 0.91597453
   0.9044806 ]
  [0.92190268 0.89046769 0.8868083  0.91020047 0.92585094 0.88768819
   0.87828466]
  [0.89487078 0.8693884  0.85757664 0.86653003 0.90359564 0.88007679
   0.87716112]]

 [[0.85301056 0.8414348  0.8673805  0.88293181 0.85568144 0.8688417
   0.88986585]
  [0.84806958 0.8108317  0.82544714 0.84720828 0.86359825 0.88581908
   0.90040623]
  [0.85604356 0.81045805 0.8047335  0.83012291 0.86524461 0.89470453
   0.87592102]
  [0.85554426 0.81356764 0.79736908 0.80259271 0.83525448 0.8843983
   0.85818303]
  [0.86421433 0.81436691 0.80029539 0.79579841 0.819411   0.86965378
   0.88410009]
  [0.88160032 0.85434114 0.81889592 0.80924773 0.85369548 0.88390842
   0.9028424 ]
  [0.89895522 0.86912624 0.83433478 0.82886494 0.87638531 0.8954713
   0.89080121]]]
```

(continues on next page)

(continued from previous page)

```
PARa absorbed per plant
[{'Hplante': [0.0, 0.0], 'parap': array([122.57339982, 154.20674688]), 'parip':␣
↪array([235.25554024, 154.20674688])}]
```

CN-Wheat results transfer

The method `to_MTG` is used to transfer results to Cn-Wheat through a MTG object with the properties `"PARa"` and `"Erel"`. The `id` argument is used with you several input scenes but you need to transfer only some of them to your MTG instance.

[23]:
```
# load a MTG instance
import os
from alinea.adel.adel_dynamic import AdelDyn
from alinea.adel.echap_leaf import echap_leaves

INPUTS_DIRPATH = os.path.join(os.path.dirname(os.path.abspath("")), "data")
adel_wheat = AdelDyn(seed=1, scene_unit="m", leaves=echap_leaves(xy_model="Soissons_
↪byleafclass"))
g = adel_wheat.load(dir=INPUTS_DIRPATH)
```

[24]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry={"scenes" : [g] })

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[25]:
```
lighting.to_MTG(mtg=g)
print(g.property("PARa"))
print(g.property("Erel"))
```

```
{19: 183.28839878423275, 34: 88.59510772768847, 813: 458.96763725505764, 814: 392.
↪72942554102866, 51: 350.64417693430573}
{19: 0.3665767975684655, 34: 0.17719021545537694, 813: 0.9179352745101153, 814: 0.
↪7854588510820574, 51: 0.7012883538686114}
```

[26]:
```
lighting = LightVegeManager(lightmodel="ratp")
lighting.build(geometry={"scenes" : [g] })

energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[27]:
```
lighting.to_MTG(mtg=g)
print(g.property("PARa"))
print(g.property("Erel"))
```

```
{19: 468.0222778320313, 34: 468.02227783203136, 813: 468.0222778320312, 814: 473.
↪66244791932104, 51: 468.02227783203125}
{19: 0.0007445579394698143, 34: 0.0007445579394698143, 813: 0.0007445579394698141, 814:␣
↪0.00030915768207336026, 51: 0.0007445579394698143}
```

2.6.6 Example of use

Here is an example with a more "realistic" canopy. We start from a single fescue and alfafa stored in .bgeom files, then we will generate copies in random positions, in order to make a canopy.

[1]:
```
import os
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
from openalea.plantgl.all import Scene
```

Canopy generation

Load the .bgeom files

[2]:
```
fet_fgeom = os.path.join(os.path.dirname(os.path.abspath("")), "data", "Fet-LD-F2.bgeom")
luz_fgeom = os.path.join(os.path.dirname(os.path.abspath("")), "data", "LD-F1.bgeom")
bgeom_files = [fet_fgeom, luz_fgeom]
bgeom_files
```

[2]:
```
['C:\\Users\\mwoussen\\cdd\\codes\\dev\\lightvegemanager\\data\\Fet-LD-F2.bgeom',
 'C:\\Users\\mwoussen\\cdd\\codes\\dev\\lightvegemanager\\data\\LD-F1.bgeom']
```

Generate copies in random position

[3]:
```
from lightvegemanager.trianglesmesh import create_heterogeneous_canopy

# scene generation parameters
nplants = 50
plant_density=130
var_plant_position=110

# generate random canopy from plant examples
if not isinstance(bgeom_files, list): bgeom_files = [bgeom_files]
scenes = []
for f in bgeom_files :
    plant_scene = Scene()
    plant_scene.read(f, 'BGEOM')

    # multiply a plant with variations
    canopy, domain = create_heterogeneous_canopy(plant_scene,
                                                 nplants=nplants,
                                                 plant_density=plant_density,
                                                 var_plant_position=var_plant_position)

    scenes.append(canopy)
```

Lighting simulation

Set simulation parameters

[4]:
```
# setup environment
environment = {}
environment["coordinates"] = [48.8 ,2.3 ,1] # latitude, longitude, timezone

# we compute only sun light in an infinite scene
environment["diffus"] = False
environment["direct"] = True
environment["reflected"] = False
environment["infinite"] = True

# CARIBU parameters
caribu_parameters = {
    "sun algo": "caribu",
    "caribu opt" : { "par": (0.10, 0.05) }
}

# inputs values for lighting
energy=500
day=264
hour=15
```

Run the simulation

[5]:
```
# Initializing the tool
lighting = LightVegeManager(lightmodel="caribu",
                                environment=environment,
                                lightmodel_parameters=caribu_parameters)



# build the scene
geometry = {"scenes" : scenes }

lighting.build(geometry)

# compute lighting
lighting.run(energy=energy, hour=hour, day=day)

# print results gathered by elements (Shapes in the plantGL Scene)
print(lighting.elements_outputs)
```

```
     Day  Hour      Organ  VegetationType       Area     par Eabs      par Ei
0    264    15  825510368               0  83.332180   76.246899   89.702234
1    264    15  825501440               0  75.958035  212.034963  249.452897
2    264    15  825503168               0   4.520565  108.134153  127.216650
3    264    15  825503824               0  57.771363   79.214402   93.193414
4    264    15  825498448               0   5.711880   97.236152  114.395473
..   ...   ...        ...             ...        ...         ...         ...
321  264    15  825485200               1   2.625990  287.633701  338.392589
```

(continues on next page)

(continued from previous page)

```
322  264    15  825485904                 1  18.000312   82.696127   97.289562
323  264    15  825486976                 1  12.152513   78.609077   92.481268
324  264    15  825488784                 1   9.200676  519.140172  610.753143
325  264    15  825489120                 1   9.200676  424.977325  499.973324

[326 rows x 7 columns]
```

[6]:
```
# visualisation
SceneWidget(lighting.plantGL_light(printtriangles=True, printvoxels=False),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 100,
            axes_helper=True)
```

[6]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'iscrYkj7jZcxvvYDWDY2BWYsh', 'data': b'x\
↪xda\x94\x9d\t|eE\x95\xff...
```

2.6.7 Misc functionnalities

Content

- Subdivision of all triangles in the scene
- Visualisation with VTK
- External tools for analysing leaf angle distribution from a mesh

Introduction

We provide more useful tools to help the lighting management

[1]:
```
import os
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
from lightvegemanager.trianglesmesh import random_triangle_generator
```

Simple mesh subdivision

If you want to refine the shadowing process in your mesh, you can subdivide all triangles.

[2]:
```
# random triangles
nb_triangles = 50
spheresize = (1., 0.3) # vertices of triangles are the sphere surface
worldsize = (0., 5.)
triangles = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for i␣
↪in range(nb_triangles)]
```

[3]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=triangles)
SceneWidget(lighting.plantGL_nolight(),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

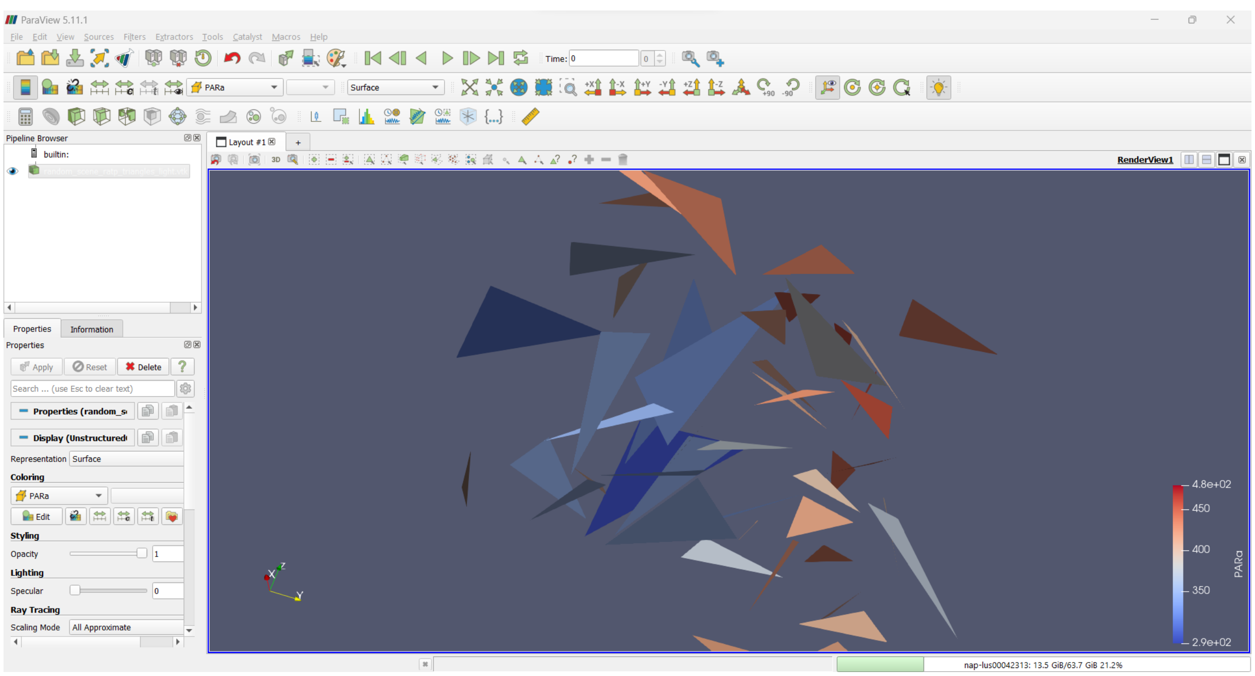
[3]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'O9xJcnvhQDpm2yuSozG1BZUoe', 'data': b'x\
↪xda\x8d\x99yXUU\x17\xc6/...
```

[4]:
```
global_scene_tesselate_level = 5
lighting.build(geometry=triangles, global_scene_tesselate_level=global_scene_tesselate_
↪level)
SceneWidget(lighting.plantGL_nolight(),
            position=(-2.5, -2.5, 0.0),
            size_display=(600, 400),
            plane=True,
            size_world = 10,
            axes_helper=True)
```

[4]:
```
SceneWidget(axes_helper=True, scenes=[{'id': 'LgksLWwDW2DzGMUPPzSXEoOnw', 'data': b'x\
↪xda\x94]\x07x]\xc5\xb16\...
```

Visualisation with VTK

PlantGL offers a good first approach to visualizing scene geometry, but software such as Paraview can take this visualization even further. LightVegeManager lets you export the scene in VTK format for further processing in ParaView.



There are three method to export VTK files: - `VTK_nolight`: exports only the geometric information, organ ID and specy ID - `VTK_light`: adds all the information about the scene's sunlight - `VTK_sun`: exports sun direction of the current iteration

Triangles exportation

[10]:
```
lighting = LightVegeManager(lightmodel="caribu")
lighting.build(geometry=triangles)
```

[11]:
```
pathfile = "random_scene"
```

[12]:
```
lighting.VTK_nolight(pathfile)
```

[13]:
```
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[14]:
```
lighting.VTK_light(pathfile)
```

Voxels exportation

If you use voxels grid, you exports them too. In our example, it will exports the triangles mesh and the voxels mesh specified in the inputs.

[15]:
```
ratp_parameters = {"voxel size" : [1., 1., 1.] }
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)
lighting.build(geometry=triangles)
```

[16]:
```
pathfile = "random_scene_ratp"
```

[17]:
```
lighting.VTK_nolight(pathfile, printtriangles=True, printvoxels=True)
```

[18]:
```
energy = 500.
hour = 15
day = 264
lighting.run(energy=energy, hour=hour, day=day)
```

[19]:
```
lighting.VTK_light(pathfile, printtriangles=True, printvoxels=True)
```

Analyze with s2v and s5

We added the possibility to call s2v and s5, two analysis tools which returns informations in order to convert the triangle mesh in a RATP grid format. Depending on the grid dimensions you specify, it will return leaf area in each voxel (depending on the barycenter position of each triangle in the grid) and leaf angle distribution.

[6]:
```
# random triangles
nb_triangles = 5
spheresize = (1., 0.3) # vertices of triangles are the sphere surface
```

(continues on next page)

(continued from previous page)

```
worldsize = (0., 5.)
triangles = [random_triangle_generator(worldsize=worldsize, spheresize=spheresize) for i␣
↪in range(nb_triangles)]
```

s5 (fortran)

[7]:
```
ratp_parameters = {"voxel size" : [1., 1., 1.] }
lighting = LightVegeManager(lightmodel="ratp", lightmodel_parameters=ratp_parameters)
lighting.build(geometry=triangles)
```

[8]:
```
lighting.s5()
```

```
--- Fin de s5.f
```

Description of `fort.60`

- xy dimension of the scene
- statistics per specy
  - Total leaf area
  - Leaf area index
  - Global zenith angle distribution
  - Global azimut angle distribution
- statistics per voxel
  - #specy #ix #iy ~iz leaf area density
  - zenith angle distribution
  - azimut angle distribution

[9]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s5", "fort.60")
with open(outfile, "r") as fichier:
    for ligne in fichier:
        print(ligne, end="")
```

```
 dimensions de la maquette (x,y):    5.000    3.000
 nombre de repetitions du motif:    1.0

 STATISTIQUES GLOBALES DE CHAQUE ESPECE

 espece:  1    surface foliaire: .411D+01    lai :  0.2738
 distribution en zenith:  0.0000 0.3407 0.0000 0.1688 0.0000 0.0000 0.3105 0.0000 0.1800
 distribution en azimuth: 0.0189 0.0000 0.3407 0.0000 0.3105 0.0000 0.0000 0.3299 0.0000


 STATISTIQUES PAR CELLULE

```

(continues on next page)

(continued from previous page)

```
  1   1   1   1       0.041
 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
  1   1   3   1       0.008
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
  1   2   3   1       0.011
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
  1   3   1   1       0.014
 0.0000 0.0000 0.0000 0.1847 0.0000 0.0000 0.0000 0.0000 0.8153
 0.1847 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.8153 0.0000
  1   3   2   1       0.180
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
  2   1   1   1       0.032
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
  2   1   2   1       0.718
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.4841 0.0000 0.5159
 0.0000 0.0000 0.0000 0.0000 0.4841 0.0000 0.0000 0.5159 0.0000
  2   1   3   1       0.027
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.9534 0.0000 0.0466
 0.0000 0.0000 0.0000 0.0000 0.9534 0.0000 0.0000 0.0466 0.0000
  2   2   3   1       0.005
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
  2   3   1   1       0.039
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
  2   3   2   1       0.104
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
  3   1   1   1       0.058
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
  3   1   2   1       0.696
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000
  3   2   3   1       0.007
 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
  3   3   3   1       0.183
 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
  4   1   2   1       0.288
 0.0000 0.0000 0.0000 0.5694 0.0000 0.0000 0.4306 0.0000 0.0000
 0.0000 0.0000 0.0000 0.0000 0.4306 0.0000 0.0000 0.5694 0.0000
  4   1   3   1       0.204
 0.0000 0.4558 0.0000 0.5442 0.0000 0.0000 0.0000 0.0000 0.0000
 0.0000 0.0000 0.4558 0.0000 0.0000 0.0000 0.0000 0.5442 0.0000
  4   2   2   1       0.157
```

(continues on next page)

(continued from previous page)

```
0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
 4   2   3   1       0.357
0.0000 0.6117 0.0000 0.3883 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.6117 0.0000 0.0000 0.0000 0.0000 0.3883 0.0000
 4   3   3   1       0.897
0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 5   1   1   1       0.034
0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 5   1   2   1       0.026
0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
 5   1   3   1       0.001
0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
 5   2   2   1       0.019
0.0000 0.0000 0.0000 1.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 1.0000 0.0000
```

Description of `leafarea`

- for each specy
  - for each voxel
    - ix | iy | iz | #specy | LAD | zenith angle distribution | azimut angle distribution

[10]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s5", "leafarea")
with open(outfile, "r") as fichier:
    for ligne in fichier:
        print(ligne, end="")
```

```
 1   1   1   1       0.041   0.000   0.000   0.000   1.000   0.000   0.000   0.000   0.000   0.000 ␣
↪1.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000
 1   1   3   1       0.008   0.000   0.000   0.000   0.000   0.000   0.000   1.000   0.000   0.000 ␣
↪0.000   0.000   0.000   0.000   1.000   0.000   0.000   0.000   0.000
 1   2   3   1       0.011   0.000   0.000   0.000   0.000   0.000   0.000   1.000   0.000   0.000 ␣
↪0.000   0.000   0.000   0.000   1.000   0.000   0.000   0.000   0.000
 1   3   1   1       0.014   0.000   0.000   0.000   0.185   0.000   0.000   0.000   0.000   0.815 ␣
↪0.185   0.000   0.000   0.000   0.000   0.000   0.000   0.815   0.000
 1   3   2   1       0.180   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   1.000 ␣
↪0.000   0.000   0.000   0.000   0.000   0.000   0.000   1.000   0.000
 2   1   1   1       0.032   0.000   0.000   0.000   0.000   0.000   0.000   0.000   0.000   1.000 ␣
↪0.000   0.000   0.000   0.000   0.000   0.000   0.000   1.000   0.000
 2   1   2   1       0.718   0.000   0.000   0.000   0.000   0.000   0.000   0.484   0.000   0.516 ␣
↪0.000   0.000   0.000   0.000   0.484   0.000   0.000   0.516   0.000
 2   1   3   1       0.027   0.000   0.000   0.000   0.000   0.000   0.000   0.953   0.000   0.047 ␣
↪0.000   0.000   0.000   0.000   0.953   0.000   0.000   0.047   0.000
 2   2   3   1       0.005   0.000   0.000   0.000   0.000   0.000   0.000   1.000   0.000   0.000 ␣
↪0.000   0.000   0.000   0.000   1.000   0.000   0.000   0.000   0.000
```

(continues on next page)

(continued from previous page)

```
  2   3   1   1       0.039  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000 ␣
↪0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000
  2   3   2   1       0.104  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000 ␣
↪0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000
  3   1   1   1       0.058  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000  0.000 ␣
↪0.000  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000
  3   1   2   1       0.696  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000  0.000 ␣
↪0.000  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000
  3   2   3   1       0.007  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000
  3   3   3   1       0.183  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000
  4   1   2   1       0.288  0.000  0.000  0.000  0.569  0.000  0.000  0.431  0.000  0.000 ␣
↪0.000  0.000  0.000  0.000  0.431  0.000  0.000  0.569  0.000
  4   1   3   1       0.204  0.000  0.456  0.000  0.544  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  0.456  0.000  0.000  0.000  0.000  0.544  0.000
  4   2   2   1       0.157  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000
  4   2   3   1       0.357  0.000  0.612  0.000  0.388  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  0.612  0.000  0.000  0.000  0.000  0.388  0.000
  4   3   3   1       0.897  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000
  5   1   1   1       0.034  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000 ␣
↪1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  5   1   2   1       0.026  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000
  5   1   3   1       0.001  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000  0.000
  5   2   2   1       0.019  0.000  0.000  0.000  1.000  0.000  0.000  0.000  0.000  0.000 ␣
↪0.000  0.000  0.000  0.000  0.000  0.000  0.000  1.000  0.000
```

s2v (c++)

[11]:
```
lighting.s2v()
```

```
--- Fin de s2v.cpp
```

Description of `s2v.log`

- Program logs
- global statistic for each specy
  - total leaf area
  - leaf area index
  - global zenith angle distribution

[12]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s2v", "s2v.log")
with open(outfile, "r") as fichier:
```

(continues on next page)

(continued from previous page)

```
    for ligne in fichier:
        print(ligne, end="")
```

```
.\s2v++.exe
Lecture du fichier parametre dans fichier :
nji=9, nja=9, njz=4
bz[0]=4
5, 5, 1, 3, 3, 1, 1
s2v.cpp:530 -> Fin de lecture de fichier
=> Calcul des distributions
==> nje rel = 1
=> Ecriture des resultats : (c|std)err, leafarea, out.dang
Il y a eu 640 depassement en z+
xl=5, yl=3, xymaille=1

STATISTIQUES GLOBALES DE CHAQUE ESPECE
esp 1 : surfT=4.10686 - Stot=4.10686, LAI=0.27379 dist d'inclinaison :0 0.340694 0 0.
↪168815 0 0 0.310531 0 0.179961
0.018865800.34069400.310531000.329910
genere le fichier out.dang => entree de sailM pour calculer la BRDF
        : xx= 0 ; Uz= 1e-009
ferrlog stream close() called.
```

Description of `s2v.can`

Copy of each triangle with the z layer in which the triangle belongs

[13]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s2v", "s2v.can")
with open(outfile, "r") as fichier:
    for ligne in fichier:
        print(ligne, end="")
```

```
p 2 100001001000 2  3 2.985657 4.113371 1.842963 2.750814 2.983446 1.871450 3.732794 3.
↪462100 2.480996
p 2 100001001000 2  3 3.433085 1.126134 1.322910 3.511158 2.888248 1.501685 1.933951 1.
↪618254 1.795654
p 2 100001001000 2  3 1.039979 3.215031 1.956095 0.182558 1.870699 2.746943 0.636121 2.
↪520374 3.280594
p 2 100001001000 2  3 2.120032 3.026543 3.191847 0.226976 3.530282 1.788605 2.980323 3.
↪379190 2.619068
p 2 100001001000 0  3 4.355116 3.219292 4.362386 4.516341 3.181239 4.363903 4.653955 2.
↪345981 3.852559
```

Description of `s2v.area`

each line contains: triangle id | z layer | triangle area

[14]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s2v", "s2v.area")
with open(outfile, "r") as fichier:
    for ligne in fichier:
        print(ligne, end="")
```

```
100001001000     2       0.615819
100001001000     2       1.39918
100001001000     2       0.739072
100001001000     2       1.27531
100001001000     0       0.0774794
```

Description of `out.dang`

File for SAIL model - line 1: global leaf area index for specy 1 - line 2: global zenith angle distribution for specy 1

[15]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s2v", "out.dang")
with open(outfile, "r") as fichier:
    for ligne in fichier:
        print(ligne, end="")
```

```
0.273790
0.000000 0.340694 0.000000 0.168815 0.000000 0.000000 0.310531 0.000000 0.179961
```

Description of `leafarea`

File for SAIL model each line: - 0 idz "Leaf area index by layer inclination class" 0 0 "total leaf area density on the z layer"

[16]:
```
outfile = os.path.join(os.path.dirname(os.path.abspath("")), "s2v", "leafarea")
with open(outfile, "r") as fichier:
    for ligne in fichier:
        print(ligne, end="")
```

```
  0   1  0.000000 0.000000 0.000000 0.353762 0.000000 0.000000 0.267263 0.000000 0.378975␣
↪0 0 0.219015
  0   2  0.000000 0.000000 0.000000 0.167022 0.000000 0.000000 0.533621 0.000000 0.299357␣
↪0 0 2.189191
  0   3  0.000000 0.823702 0.000000 0.147280 0.000000 0.000000 0.028594 0.000000 0.000425␣
↪0 0 1.698650
  0   4  0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000␣
↪0 0 0.000000
```

2.6.8 More examples

[1]:
```
import openalea.plantgl.all as pgl_all
from lightvegemanager.LVM import LightVegeManager
from pgljupyter import SceneWidget
```

Sun positions

The `print_sun` function allows you to compare the sun position differences between RATP and CARIBU internal computation.

[38]:
```
from lightvegemanager.sun import print_sun

print(print_sun.__doc__)
```

```
Prints sun position ouputs from RATP and CARIBU algorithm with the same inputs

    :param day: input day
    :type day: int
    :param hour: input hour
    :type hour: int
    :param coordinates: [latitude, longitude, timezone]
    :type coordinates: list
    :param truesolartime: activates true solar time or local time to compute sun position
    :type truesolartime: bool
```

[11]:
```
print_sun(201, 12, [0.0, 0.0, 0.0], True)
print_sun(201, 12, [40.0, 0.0, 0.0], True)
print_sun(201, 16, [40.0, 12.0, 2.0], True)
print_sun(201, 16, [40.0, 12.0, 2.0], False)
print_sun(347, 16, [46.0, 0.0, 0.0], True)
```

```
---      SUN COORDONATES         ---
--- Convention x+ = North, vector from sky to floor
--- azimut: south clockwise E = -90° W = 90°     zenith:
↪zenith = 0° horizon = 90°
--- day: 201     hour: 12        latitude: 0.00 °
--- true solar time
--- RATP ---
         azimut: 180.000         zenith: 69.227
         x: -0.354671    y: -0.000000    z: -0.934991
--- CARIBU ---
         azimut: 180.000         zenith: 68.944
         x: -0.359285    y: -0.000000    z: -0.933228


---      SUN COORDONATES         ---
--- Convention x+ = North, vector from sky to floor
--- azimut: south clockwise E = -90° W = 90°     zenith:
↪zenith = 0° horizon = 90°
--- day: 201     hour: 12        latitude: 40.00 °
--- true solar time
```

(continues on next page)

(continued from previous page)

```
--- RATP ---
         azimut: 0.000   zenith: 70.773
         x: 0.329307     y: -0.000000    z: -0.944223
--- CARIBU ---
         azimut: 0.000   zenith: 71.148
         x: 0.323132     y: -0.000000    z: -0.946354


---      SUN COORDONATES         ---
--- Convention x+ = North, vector from sky to floor
--- azimut: south clockwise E = -90° W = 90°     zenith: ␣
↪zenith = 0° horizon = 90°
--- day: 201     hour: 16        latitude: 40.00 °
--- true solar time
--- RATP ---
         azimut: 87.963          zenith: 35.881
         x: 0.028807     y: -0.809726    z: -0.586100
--- CARIBU ---
         azimut: 88.295          zenith: 36.115
         x: 0.024030     y: -0.807482    z: -0.589402


---      SUN COORDONATES         ---
--- Convention x+ = North, vector from sky to floor
--- azimut: south clockwise E = -90° W = 90°     zenith: ␣
↪zenith = 0° horizon = 90°
--- day: 201     hour: 16        latitude: 40.00 °
--- local time, true solar time is 18.70
--- RATP ---
         azimut: 112.480         zenith: 5.641
         x: -0.380514    y: -0.919537    z: -0.098291
--- CARIBU ---
         azimut: 112.736         zenith: 5.895
         x: -0.384436    y: -0.917420    z: -0.102706


---      SUN COORDONATES         ---
--- Convention x+ = North, vector from sky to floor
--- azimut: south clockwise E = -90° W = 90°     zenith: ␣
↪zenith = 0° horizon = 90°
--- day: 347     hour: 16        latitude: 46.00 °
--- true solar time
--- RATP ---
         azimut: 52.853          zenith: 2.129
         x: 0.603439     y: -0.796543    z: -0.037150
--- CARIBU ---
         azimut: 53.169          zenith: 2.592
         x: 0.598849     y: -0.799584    z: -0.045228
```

Two planes

A small example with two horizontal planes, in order to check if the xy orientation according to North-East-South-West.

[2]:
```
geom1 = pgl_all.FaceSet([(0,0,0),(2,0,0), (2,2,0),(0,2,0)],[range(4)]) # plaque dessous
geom2 = pgl_all.FaceSet([(0,1,1),(2,1,1), (2,3,1),(0,3,1)],[range(4)]) # plaque dessus,␣
↪décalée en y (axe est-ouest) vers l'ouest
s = pgl_all.Scene([pgl_all.Shape(geom1, pgl_all.Material((250,0,0),1), 888),
                    pgl_all.Shape(geom2, pgl_all.Material((0,250,0),1), 999)])
```

[3]:
```
# input dict
geometry = {}
environment = {}
caribu_parameters = {}

# Paramètres pré-simulation
geometry["scenes"] = [s]

environment["coordinates"] = [0. ,0. ,0.] # latitude, longitude, timezone
environment["direct"] = True
environment["diffus"] = False
environment["reflected"] = False
environment["caribu opt"] = {}
environment["caribu opt"]["par"] = (0.10, ) # plaques opaques
environment["infinite"] = False

## Paramètres CARIBU ##
caribu_parameters["sun algo"] = "caribu"
caribu_parameters["caribu opt"] = {}
caribu_parameters["caribu opt"]["par"] = (0.10, ) # plaques opaques

# Déclaration de l'objet
lghtcaribu = LightVegeManager(environment=environment,
                                lightmodel="caribu",
                                lightmodel_parameters=caribu_parameters)
# création de la scène dans l'objet
lghtcaribu.build(geometry, global_scene_tesselate_level=5)
```

[39]:
```
# forçage arbitraire en µmol.m-2.s-1
PARi = 500
# jour arbitraire (21 septembre)
day = 264

# heure matin, soleil arrive de l'est, les 2 plaques reçoivent tout le rayonnement
hour = 16
print("--- day %i, hour %i"%(day, hour))

# Calcul du rayonnement
lghtcaribu.run(energy=PARi, day=day, hour=hour, parunit="micromol.m-2.s-1",␣
↪truesolartime=True)
```

```
--- day 264, hour 16
```

[10]:
```
# visualisation
SceneWidget(lghtcaribu.plantGL_light(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=False,
            size_world = 10,
            axes_helper=True)
```

[10]:
```
SceneWidget(axes_helper=True, plane=False, scenes=[{'id': 'PidZp3tokVAgJYlbO4DHRXebB',
↪'data': b'x\xda\x9c\xbd...
```

Examples with RATP and stem elements

```
Stems processing checking
-------------------------

Scene for verification is a set of horizontal planes in one voxel as so:

-------------
|   ----    |
|     ____  |
| ----      |
-------------

We compare the PAR mean on all the planes between CARIBU and RATP

Stems processing
    * CARIBU
        Computing of incident PAR is done with no transmittance, triangles are opaques

    * RATP
        triangles area are divided by 2
```

[34]:
```
geom1 = pgl_all.FaceSet([(0.1,0.1,0.96),(0.1,0.6,0.96), (0.6,0.6,0.96),(0.6,0.1,0.96)],
↪[range(4)])
geom2 = pgl_all.FaceSet([(0.1,0.5,0.806),(0.1,0.9,0.806), (0.6,0.9,0.806),(0.6,0.5,0.
↪806)],[range(4)])
geom3 = pgl_all.FaceSet([(0.5,0.1,0.646),(0.5,0.7,0.646), (0.9,0.7,0.646),(0.9,0.1,0.
↪646)],[range(4)])
geom4 = pgl_all.FaceSet([(0.3,0.5,0.486),(0.3,0.9,0.486), (0.8,0.9,0.486),(0.8,0.5,0.
↪486)],[range(4)])
geom5 = pgl_all.FaceSet([(0.3,0.1,0.32),(0.3,0.5,0.32), (0.8,0.5,0.32),(0.8,0.1,0.32)],
↪[range(4)])
geom6 = pgl_all.FaceSet([(0.2,0.4,0.16),(0.2,0.9,0.16), (0.6,0.9,0.16),(0.6,0.4,0.16)],
↪[range(4)])
s = pgl_all.Scene([pgl_all.Shape(geom1, pgl_all.Material((250,0,0),1), 888),
                   pgl_all.Shape(geom2, pgl_all.Material((250,0,0),1), 888),
                   pgl_all.Shape(geom3, pgl_all.Material((250,0,0),1), 888),
                   pgl_all.Shape(geom4, pgl_all.Material((250,0,0),1), 888),
                   pgl_all.Shape(geom5, pgl_all.Material((250,0,0),1), 888),
                   pgl_all.Shape(geom6, pgl_all.Material((250,0,0),1), 888)])
```

[35]:
```
geometry = {}
environment = {}
ratp_parameters = {}
caribu_parameters = {}

# Paramètres pré-simulation
geometry["scenes"] = [s]
geometry["stems id"] = [(888, 0)]

environment["coordinates"] = [0., 0., 0.] # latitude, longitude, timezone
environment["sky"] = "turtle46" # turtle à 46 directions par défaut
environment["diffus"] = False
environment["direct"] = True
environment["reflected"] = False
environment["infinite"] = False

## Paramètres CARIBU ##
caribu_parameters["sun algo"] = "caribu"
caribu_parameters["caribu opt"] = {}
caribu_parameters["caribu opt"]["par"] = (0.10, 0.05)
lghtcaribu = LightVegeManager(environment=environment,
                                lightmodel="caribu",
                                lightmodel_parameters=caribu_parameters)
lghtcaribu.build(geometry)
```

[36]:
```
## Paramètres RATP ##
dv = 1. # m
dx, dy, dz = dv, dv, dv # m
ratp_parameters["voxel size"] = [dx, dy, dz]
ratp_parameters["soil reflectance"] = [0., 0.]
ratp_parameters["mu"] = [1.]
ratp_parameters["tesselation level"] = 0
ratp_parameters["angle distrib algo"] = "compute global"
ratp_parameters["nb angle classes"] = 30
ratp_parameters["reflectance coefficients"] = [[0.1, 0.05]]
lghtratp = LightVegeManager(environment=environment,
                            lightmodel="ratp",
                            lightmodel_parameters=ratp_parameters)
lghtratp.build(geometry)
```

[28]:
```
# visualisation
SceneWidget(lghtratp.plantGL_nolight(printvoxels=True),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=False,
            size_world = 2,
            axes_helper=True)
```

[28]:
```
SceneWidget(axes_helper=True, plane=False, scenes=[{'id': '31qAWA8Lxo9AjwlYbU2IJwjFk',
↪'data': b'x\xda\x85\x95...
```

[29]:
```
PARi=500
```

(continues on next page)

(continued from previous page)

```
day=100.
hour=12
```

[ ]:
```
PARi=500
day=100.
hour=17.5
```

[31]:
```
lghtcaribu.run(energy=PARi, day=day, hour=hour, parunit="micromol.m-2.s-1",␣
↪truesolartime=True)
print("=== CARIBU ===")
print(lghtcaribu.elements_outputs)
# visualisation
SceneWidget(lghtcaribu.plantGL_light(),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=False,
            size_world = 2,
            axes_helper=True)
```

```
=== CARIBU ===
     Day  Hour  Organ  VegetationType  Area    par Eabs      par Ei
0  100.0    12    888               0  1.29  226.615845  251.795383
```

[31]:
```
SceneWidget(axes_helper=True, plane=False, scenes=[{'id': 'euPE10Vd7GkTxzNRtZPTE36Dx',
↪'data': b'x\xda\x85\xd5...
```

[33]:
```
lghtratp.run(energy=PARi, day=day, hour=hour, parunit="micromol.m-2.s-1",␣
↪truesolartime=True)
print("=== RATP ===")
print(lghtratp.elements_outputs)
# visualisation
SceneWidget(lghtratp.plantGL_light(printvoxels=True),
            position=(0.0, 0.0, 0.0),
            size_display=(600, 400),
            plane=False,
            size_world = 2,
            axes_helper=True)
```

```
=== RATP ===
     Day  Hour  Organ  VegetationType  Area        PARa  Intercepted  \
0  100.0  12.0    888               2  1.29  377.546967     0.487036

   Transmitted   SunlitPAR  SunlitArea  ShadedPAR  ShadedArea
0     0.487036  499.828644    0.487203        0.0    0.157797
```

[33]:
```
SceneWidget(axes_helper=True, plane=False, scenes=[{'id': 'GNw9osvivwBapMiJRL2W1wlXa',
↪'data': b'x\xda\x85\xd6...
```

2.7 Reference Guide

This manual details, for each module of *lightvegemanager*, the functions and objects included in *lightvegemanager*, describing what they are and what they do.

Contents

2.7.1 `LVM` module

LightVegeManager

Main class of the tool. Calls all the other modules in `src`.

3 inputs dict for setting all parameters:

```
geometry = {
            "scenes" : [scene0, scene1, scene2, ...] ,
            "domain" : ((xmin, ymin), (xmax, ymax)),
            "stems id" : [(id_element, id_scene), ...],
            "transformations" : {
                                    "scenes unit" : kwarg ,
                                    "rescale" : kwarg ,
                                    "translate" : kwarg ,
                                    "xyz orientation" : kwarg
                                    }
                }
```

```
environment = {
                "coordinates" : [latitude, longitude, timezone] ,

                "sky" : "turtle46" ,
                "sky" : ["file", filepath] ,
                "sky" : [nb_azimut, nb_zenith, "soc" or "uoc"] ,

                "direct" : bool, # sun radiations
                "diffus" : bool, # sky radiations
                "reflected" : bool, # reflected radiation in the canopy
                "infinite" : bool, # infinitisation of the scene
                }
```

Currently LightVegeManager handles the light models RATP and CARIBU:

```
caribu_args = {
                "sun algo" : "ratp",
                "sun algo" : "caribu",

                    "caribu opt" : {
                                    band0 = (reflectance, transmittance),
                                    band1 = (reflectance, transmittance),
                                    ...
                                    },
```

(continues on next page)

(continued from previous page)

```
                    "debug" : bool,
                    "soil mesh" : bool,
                    "sensors" : ["grid", dxyz, nxyz, orig, vtkpath, "vtk"]
                }
```

```
ratp_args = {
                # Grid specifications
                "voxel size" : [dx, dy, dz],
                "voxel size" : "dynamic",

                "origin" : [xorigin, yorigin, zorigin],
                "origin" : [xorigin, yorigin],

                "number voxels" : [nx, ny, nz],
                "grid slicing" : "ground = 0."
                "tesselation level" : int

                # Leaf angle distribution
                "angle distrib algo" : "compute global",
                "angle distrib algo" : "compute voxel",
                "angle distrib algo" : "file",

                "nb angle classes" : int,
                "angle distrib file" : filepath,

                # Vegetation type
                "soil reflectance" : [reflectance_band0, reflectance_band1, ...],
                "reflectance coefficients" : [reflectance_band0, reflectance_band1, ...],
                "mu" : [mu_scene0, mu_scene1, ...]
            }
```

See also:

For more details *Inputs description*

class `LVM.`**`LightVegeManager`**(*environment={}, lightmodel='', lightmodel_parameters={}, main_unit='m'*)

Bases: `object`

Main class for the tool LightVegeManager

Common simulation order:

`input geometries -> build and prepare data -> call light model -> transfer results to plant models`

It includes:

Main methods:

- `__init__`: initializes and builds static object for the rest of simulation
- *`build()`*: builds and prepare all geometric meshes
- *`run()`*: calls a light model and manages its inputs and outputs

Transfer methods:

- *`to_MTG()`*: transfers ligthing results to a MTG table

- *to_l_egume()*: transfers ligthing results to l-egume by creating two arrays as inputs for the plant model

Analysis tools: analyses a set of triangles and formats them as a turbid medium inputs

- *s5()*: fortran tool
- *s2v()*: c++ tool

Visualisation tools:

- *plantGL_nolight()*: return a plantGL scene from the geometry
- *plantGL_light()*: return a plantGL scene from the geometry with lighting results
- *plantGL_sensors()*: return a plantGL scene from virtual sensors if created
- *VTK_nolight()*: write VTK file with only geometric informations
- *VTK_light()*: write VTK file with geometric informations and associated light results
- *VTK_sun()*: write VTK file representing the sun as a line

Getters: to use light results with external routines

- *riri5_transmitted_light()*
- *riri5_intercepted_light()*
- *elements_outputs()*
- *triangles_outputs()*
- *voxels_outputs()*
- *sensors_outputs()*
- *sun()*
- *soilenergy()*
- *maxtrianglearea()*
- *legume_empty_layers()*
- *tesselationtime()*
- *modelruntime()*
- *leafangledistribution()*

Parameters

- **environment** (*dict, optional*) – Environment parameters, defaults to {}
- **lightmodel** (*str, optional*) – either `"ratp"` or `"caribu"`, defaults to “”
- **lightmodel_parameters** (*dict, optional*) – light model parameters, defaults to {}
- **main_unit** (*str, optional*) – measure unit for the global scene where the light will be computed, defaults to “m”

Raises

ValueError – lightmodel entry not valid, either `'ratp'` or `'caribu'`

build(*geometry={}*, *global_scene_tesselate_level=0*)

Builds a mesh of the simulation scene in the right light model format

Parameters

- **geometry** (*dict, optional*) – geometric parameters, contains geometric scenes, defaults to {}
- **global_scene_tesselate_level** (*int, optional*) – option to subdivide all triangles of the mesh a certain number of times (to fine tuning the mesh), defaults to 0

Raises

`ValueError` – Currently, converting voxels mesh to triangles mesh is not possible

run(*energy=0.0*, *day=0*, *hour=0*, *parunit='micromol.m-2.s-1'*, *truesolartime=False*, *id_sensors=None*)

Calls the light model and formats lighting results

Parameters

- **energy** (*float, optional*) – input radiation energy, defaults to 0
- **day** (*int, optional*) – simulation day, defaults to 0
- **hour** (*int, optional*) – simulation hour, defaults to 0
- **parunit** (*str, optional*) – input energy unit, light models manages radiations in different unit, you can precise input unit and LightVegeManager will convert it in the right unit, defaults to "micromol.m-2.s-1"
- **truesolartime** (*bool, optional*) – simulation hour is a true solar time or local time (depending on simulation coordinates), defaults to False
- **id_sensors** (*list, optional*) – if you use CARIBU with a grid of virtual sensors, you have to precise which input scenes the grid must match, defaults to None

Raises

- `ValueError` – with CARIBU you can precise the sun algorithm to calculate sun position, can be either `"caribu"` or `"ratp"`
- `ValueError` – valid radiations are `"direct"`, `"diffuse"`, `"reflected"`

to_MTG(*energy=1.0*, *mtg=None*, *id=None*)

Transfers lighting results to a MTG table.

> **Warning:** The *run()* must have been called before to have results dataframes.

Results are `pandas.Dataframe` stored in the `self`

Parameters

- **energy** (*float, optional*) – input energy, defaults to 1
- **mtg** (*MTG, optional*) – MTG table with `"PARa"` and `"Erel"` entries in its properties, defaults to None
- **id** (*list or tuple, optional*) – you can precise to which input scenes the MTG table corresponds, defaults to None

Raises

`AttributeError` – you need to call :func:run first

to_l_egume(*energy=1.0*, *m_lais=[]*, *list_lstring=[]*, *list_dicFeuilBilanR=[]*, *list_invar=[]*, *id=None*)

Transfers lighting results to l-egume

> **Warning:** The *run()* must have been called before to have results dataframes.

> **Warning:** l-egume needs transmitted energy informations located in a grid of voxels. You need to have the same dimensions in the lighting results.
>
> - With RATP, RATP grid must have the same dimensions as l-egume intern grid.
> - With CARIBU, you must create a grid of virtual sensors in the same dimensions as l-egume intern grid.

Results are `pandas.Dataframe` stored in the `self`

Parameters

- **energy** (*float, optional*) – input energy, defaults to 1
- **m_lais** (*numpy.array, optional*) – leaf area represented in a numpy.array of dimension [number of species, number of z layers, number of y layers, number of x layers], defaults to []
- **list_lstring** (*list of dict, optional*) – from l-egume, each element corresponds to an input specy of l-egume. Each element is a dict lstring stores the l-system of each plant, defaults to []
- **list_dicFeuilBilanR** (*list of dict, optional*) – from l-egume, each element corresponds to an input specy of l-egume. Each element is a dict dicFeuiBilanR stores correspondances between voxels grid and each plant, defaults to []
- **list_invar** (*list of dict, optional*) – from l-egume, each element corresponds to an input specy of l-egume. Each element is a dict invar stores instant intern variables of l-egume., defaults to []
- **id** (*list, optional*) – list of indices from input scenes which corresponds to current l-egume instance. If you have several plantmodel among the input scenes, you need to precise which results you want to transfer to which instance of l-egume. defaults to None

Raises

- **AttributeError** – you need to call *run()* first
- **ValueError** – unknown light model

Returns

if light model is RATP `transfer_ratp_legume()` :

- `res_abs_i`: absorbed energy in each voxels of a grid matching the dimensions of l-egume intern grid of voxels. One value for each specy.
- `res_trans`: transmitted energy in each voxels of a grid matching the dimensions of l-egume intern grid of voxels

if light model is CARIBU :func:transfer_caribu_legume :

- update of list_invar: updates the keys `"parap"` and `"parip"` for each specy. Cumulatative energy per plant

- `res_trans`: transmitted energy in each voxels of a grid matching the dimensions of l-egume intern grid of voxels

Return type
numpy.array

`s5()`

Creates inputs files for s5 and runs it s5 is an external tool made to analyse a set of triangles in order to use a grid of voxels. It also computes leaf angle distribution from the triangulation

Note: All files are located in `s5` folder

Input files created

- fort.51: contains triangulation. Possibility to precise stem elements it is registered in the instance of LightVegeManager
- s5.par: stores grid of voxels informations and number of entities

Output files created

- fort.60

  dimensions xy of the grid

  stats by specy

  - total leaf area
  - leaf area index
  - global zenital leaf angle distribution
  - global azimutal leaf angle distribution

  stats by voxels

  - #specy | #ix | #iy | #iz (coordinate xyz of the voxel) | eaf area density
  - local zenital leaf angle distribution
  - local azimutal leaf angle distribution

- leafarea: for each specy, for each voxel

  ix | iy | iz | #specy | LAD | zenital-distribution | azimutal-distribution

example

```
>>> myscene # a plantgl Scene
>>> testofs5 = LightVegeManager() # create a instance
>>> testofs5.build( geometry={ "scenes" : [myscene] } ) # build the geometry
>>> testofs5.s5() # run of s5, creates input and output files
```

`s2v()`

Creates inputs files for s2v and runs it s5 is an external tool made to analyse a set of triangles in order to use a grid of voxels. It also computes leaf angle distribution from the triangulation

Note: All files are located in `s2v` folder

Input files created

- fort.51: stores triangulation. Possibility to precise stem elements it is registered in the instance of LightVegeManager
- s2v.par: stores grid of voxels informations and number of entities

Output files created

- s2v.log

  logs about processing

  global statistics

  – total leaf area per specy

  – total leaf area

  – leaf area index

  – global zenital leaf angle distribution

- s2v.can

  z layer where each triangle is located (and copy its vertices)

- s2v.area

  triangle id | z layer | triangle area

- out.dang: SAIL file

  line 1: global leaf area index for specy 1 line 2: global zenital leaf angle distribution for specy 1

- leafarea: SAIL file

  each line: 0 | idz | leaf area index for each slope class in current z layer | 0 | 0 | leaf area density on the layer

example

```
>>> myscene # a plantgl Scene
>>> testofs2v = LightVegeManager() # create a instance
>>> testofs2v.build( geometry={ "scenes" : [myscene] } ) # build the geometry
>>> testofs2v.s2v() # run of s2v, creates input and output files
```

VTK_nolight(*path*, *i=None*, *printtriangles=True*, *printvoxels=True*)

Writes a VTK from mesh(es) in `self`, with only geometric informations

Parameters

- **path** (*string*) – file and path name
- **i** (*int, optional*) – associate the created file with an indice in its filename, defaults to None
- **printtriangles** (*bool, optional*) – write triangulation if one has been created in :func:build, defaults to True
- **printvoxels** (*bool, optional*) – write grid of voxels if one has been created in :func:build, defaults to True

VTK_light(*path*, *i=None*, *printtriangles=True*, *printvoxels=True*)

Writes a VTK from mesh(es) in `self`, with geometric informations and lighting results

Warning: The `run()` must have been called before to have results dataframes.

Parameters

- **path** (*string*) – file and path name
- **i** (*int, optional*) – associate the created file with an indice in its filename, defaults to None
- **printtriangles** (*bool, optional*) – write triangulation if one has been created in `build()`, defaults to True
- **printvoxels** (*bool, optional*) – write grid of voxels if one has been created in `build()`, defaults to True

Raises

`AttributeError` – you need to call `run()` first

VTK_sun(*path*, *scale=2*, *orig=(0, 0, 0)*, *center=True*, *i=None*)

Write a VTK file representing the sun by a simple line

Warning: The `run()` must have been called before to have results dataframes.

if center is False, orig is the starting point the line and it ends at orig + scale*sun.position

if center is True, orig is the middle point of the line.

Parameters

- **path** (*string*) – file and path name
- **scale** (*int, optional*) – size of the line, defaults to 2
- **orig** (*tuple, optional*) – starting or middle point of the line, defaults to (0,0,0)
- **center** (*bool, optional*) – if True orig is the middle of the line, otherwise it is the starting point, defaults to True
- **i** (*int, optional*) – associate the created file with an indice in its filename, defaults to None

Raises

`AttributeError` – you need to call `run()` first

plantGL_sensors(*light=False*)

plantGL_nolight(*printtriangles=True*, *printvoxels=False*)

Return a plantGL Scene from mesh(es) in `self`, with geometric informations

Parameters

- **printtriangles** (*bool, optional*) – write triangulation if one has been created in `build()`, defaults to True
- **printvoxels** (*bool, optional*) – write grid of voxels if one has been created in `build()`, defaults to True

`plantGL_light`(*printtriangles=True*, *printvoxels=False*)

Return a plantGL Scene from mesh(es) in `self`, with geometric informations and lighting results

> **Warning:** The `run()` must have been called before to have results dataframes.

Parameters

- **printtriangles** (*bool, optional*) – write triangulation if one has been created in `build()`, defaults to True
- **printvoxels** (*bool, optional*) – write grid of voxels if one has been created in `build()`, defaults to True

Raises

`AttributeError` – you need to call `run()` first

property riri5_transmitted_light

transmitted results if light model is RiRi light

Returns

transmitted energy following a grid of voxels

Return type

numpy.array

property riri5_intercepted_light

Intercepted results if light model is RiRi light,

Returns

intercepted energy following a grid of voxels, for each specy

Return type

numpy.array

property elements_outputs

Lighting results aggregate by element

Returns

Column names can change depending on the light model. Commonly, there is element indice, its area and intercepted energy

Return type

pandas.Dataframe

property triangles_outputs

Lighting results aggregate by triangle, if it has a triangulation in its inputs

Returns

See also:

`outputs` for column names

Return type

pandas.Dataframe

property voxels_outputs

Lighting results aggregate by voxels, only with RATP as the selected light model

Returns

See also:

outputs for column names

Return type
pandas.Dataframe

property sensors_outputs

Lighting results aggregate by sensors. Only with CARIBU if you activated the virtual sensors option

Returns
The output format is the same as CARIBU output format. Dict with `Eabs` key for absorbed energy and `Ei` for incoming energy

Return type
dict

property sun

Return sun position of the last :func:run call

Returns
vector (x, y, z)

Return type
tuple

property soilenergy

Return soil energy, only with CARIBU if soilmesh option is activated

Returns
The output format is the same as CARIBU output format. Dict with `Eabs` key for absorbed energy and `Ei` for incoming energy

Return type
dict

property maxtrianglearea

Returns the largest triangle of triangles mesh Computed in *build()*

Returns
area the triangle

Return type
float

property legume_empty_layers

Returns number of empty layers between top of the canopy and number of z layers expected by l-egume

Returns
result of `fill_ratpgrid_from_legumescene()`

Return type
int

property tesselationtime

summary

Returns
description

Return type
type

property modelruntime
Returns running time of the light model

Returns
time in s

Return type
float

property leafangledistribution

property domain
Returns the largest triangle of triangles mesh Computed in *build()*

Returns
area the triangle

Return type
float

2.7.2 CARIBUinputs module

CARIBUinputs

Manages and prepares input information for CARIBU.

In this module we use two of the LightVegeManager's inputs dict.

- `parameters` corresponding to CARIBU parameters and contains

```
caribu_args = {
                "sun algo" : "ratp",
                "sun algo" : "caribu",

                "caribu opt" : {
                                band0 = (reflectance, transmittance),
                                band1 = (reflectance, transmittance),
                                ...
                                },
                "debug" : bool,
                "soil mesh" : bool,
                "sensors" : ["grid", dxyz, nxyz, orig, vtkpath, "vtk"]
                }
```

- `geometry` corresponding to the geometric information with the scenes inputs

```
geometry = {
            "scenes" : [scene0, scene1, scene2, ...] ,
            "domain" : ((xmin, ymin), (xmax, ymax)),
            "stems id" : [(id_element, id_scene), ...],
            "transformations" : {
                                    "scenes unit" : kwarg ,
                                    "rescale" : kwarg ,
```

(continues on next page)

(continued from previous page)

```
                                                    "translate" : kwarg ,
                                                    "xyz orientation" : kwarg
                                                }
                        }
```

See also:

For more details *Inputs description*

`CARIBUinputs.`**`Prepare_CARIBU`**(*trimesh*, *geometry*, *matching_ids*, *minmax*, *parameters*, *infinite*, *idsensors*)

Format optical parameters and prepare virtual sensors and debug if activated

Parameters

- **`trimesh`** (*`dict`*) – triangles mesh aggregated by indice elements

```
{ id : [triangle1, triangle2, ...]}
```

- **`geometry`** (*`dict`*) – geometric parameters from inputs of LightVegeManager
- **`matching_ids`** (*`dict`*) – dict that matches new element indices in trimesh with specy indice and input element indice,

```
matching_ids = { new_element_id : (input_element_id, specy_id)}
```

- **`minmax`** (*`[3-tuple, 3-tuple]`*) – list of mininuml point and maximum point in a triangles mesh
- **`parameters`** (*`dict`*) – RATP parameters from inputs of LightVegeManager
- **`infinite`** (*`bool`*) – if the user wishes to activate infinitisation of the grid
- **`idsensors`** (*`list of int`*) – Sets which input scenes the grid of virtual sensors will match. Elements of this list refers to indices in the list `geometry["scenes"]`. If no input indices are given, the grid will match the global scene `trimesh`. Otherwise, you can match the grid to a specific set of input scenes

Returns

- **`opt`: optical parameters formatted for CARIBU. It takes the form of dict, where each** key is a bandlength (PAR, NIR etc…) and values are transmittance and reflectance (or only reflectance for stems elements)
- **`sensors_caribu`: geometric data of the virtual sensors in CARIBU scene format. Sensors** are horizontal square made of two triangles

```
sensors_caribu = { sensor_id : [triangle1, triangle2], ...}
```

- `debug`: boolean if user wants to activate th edebug option in CARIBU

Return type

dict, dict, bool

`CARIBUinputs.`**`CARIBU_opticals`**(*matching_ids*, *parameters*, *stems_id=None*)

Sets optical parameters for CARIBU from LightVegeManager inputs It removes transmittance for stems elements if precised

Parameters

- **matching_ids** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, `matching_ids = { new_element_id : (input_element_id, specy_id)}`
- **parameters** (*dict*) – RATP parameters from inputs of LightVegeManager
- **stems_id** (*list of 2-tuple, optional*) – list of potential stems element in the input scenes, defaults to None

Returns
optical parameters formatted for CARIBU. It takes the form of dict, where each key is a bandlength (PAR, NIR etc. . .) and values are transmittance and reflectance (or only reflectance for stems elements)

Return type
dict

`CARIBUinputs.`**`create_caribu_legume_sensors`**(*dxyz*, *nxyz*, *orig*, *pmax*, *trimesh*, *matching_ids*, *id_sensors*, *infinite*)

Creates a set of virtual sensors following a voxels grid each sensor is a square made by two triangles and takes place on the bottom face of a voxel The grid follow the xyz axis (and so the voxels)

Parameters

- **dxyz** (*list*) – [dx, dy, dz] size of a voxel in each direction
- **nxyz** (*list*) – [nx, ny, nz] number of voxels on each axis
- **orig** (*list (or tuple)*) – [x_origin, y_origin, z_origin], origin of the grid, cartesian coordinates
- **pmax** (*lsit (or tuple)*) – [x, y, z] maximum point of trimesh in the xyz space
- **trimesh** (*dict*) – triangles mesh aggregated by indice elements

```
{ id : [triangle1, triangle2, ...]}
```

- **matching_ids** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, `matching_ids = { new_element_id : (input_element_id, specy_id)}`
- **idsensors** (*list of int*) – Sets which input scenes the grid of virtual sensors will match. Elements of this list refers to indices in the list `geometry["scenes"]`. If no input indices are given, the grid will match the global scene `trimesh`. Otherwise, you can match the grid to a specific set of input scenes
- **infinite** (*bool*) – if the user wishes to activate infinitisation of the grid

Returns

it returns 3 objects:

- **sensors_caribu, geometric data of the virtual sensors in CARIBU scene format. Sensors**
  are horizontal square made of two triangles `sensors_caribu = { sensor_id : [triangle1, triangle2], ...}`
- `s_capt`, PlantGL scene of the virtual sensors
- **sensors_maxcenter, [x, y, z] representing point on the highest sensors layer and middle of**
  xy plane in the grid

Return type
dict, PlantGL scene, list

`CARIBUinputs.run_caribu`(*c_scene*, *direct_active*, *infinite*, *sensors*, *energy=1.0*)
runs caribu depending on input options

Parameters

- **c_scene** (*CaribuScene*) – instance of CaribuScene containing geometry, light source(s), opt etc…
- **direct_active** (*bool*) – Whether only first order interception is to be computed, default is True (no rediffusions)
- **infinite** (*bool*) – if the user wishes to activate infinitisation of the grid
- **sensors** (*dict*) – geometric data of the virtual sensors in CARIBU scene format. Sensors are horizontal square made of two triangles

```
sensors_caribu = { sensor_id : [triangle1, triangle2], ...}
```

- **energy** (*float, optional default is 1.*) – input energy value

Returns

results are stored in two dict

- **raw (dict of dict) a {band_name: {result_name: property}} dict of dict.**
  Except for result_name='sensors', each property is a {primitive_id: [values,]} dict containing results for individual triangles of the primitive
- **aggregated (dict of dict)**
  [a {band_name: {result_name: property}}] Except for result_name='sensors', each property is a {primitive_id: value} dict containing aggregated results for each primitive

result_name are :

- area (float): the individual areas (m2)
- Eabs (float): the surfacic density of energy absorbed (m-2)
- Ei (float): the surfacic density of energy incoming (m-2) additionally, if split_face is True
- Ei_inf (float): the surfacic density of energy incoming on the inferior face (m-2)
- Ei_sup (float): the surfacic density of energy incoming on the superior face (m-2)
- sensors (dict of dict): area, surfacic density of incoming direct energy and surfacic density of incoming total energy of sensors grouped by id, if any

Return type
dict of dict, dict of dict

2.7.3 `RATPinputs` module

RATPinputs

Manage vegetation and meteo input informations for RATP

The argument *parameters* refers to one the three inputs dict of LightVegeManager. It is structured as so:

```
ratp_args = {
        # Grid specifications
        "voxel size" : [dx, dy, dz],
        "voxel size" : "dynamic",

        "origin" : [xorigin, yorigin, zorigin],
        "origin" : [xorigin, yorigin],

        "number voxels" : [nx, ny, nz],
        "grid slicing" : "ground = 0.",
        "tesselation level" : int,

        "full grid" : bool,

        # Leaf angle distribution
        "angle distrib algo" : "compute global",
        "angle distrib algo" : "compute voxel",
        "angle distrib algo" : "file",

        "nb angle classes" : int,
        "angle distrib file" : filepath,

        # Vegetation type
        "soil reflectance" : [reflectance_band0, reflectance_band1, ...],
        "reflectance coefficients" : [reflectance_band0, reflectance_band1, ...],
        "mu" : [mu_scene0, mu_scene1, ...]
    }
```

See also:

Inputs description

`RATPinputs.`**`RATP_vegetation`**(*parameters*, *angle_distrib*, *reflected*)

Initialise a RATP Vegetation object from LightVegeManager input datas

Parameters

- **`parameters`** (*`dict`*) – RATP parameters from inputs of LightVegeManager
- **`angle_distrib`** (*`dict`*) – leaf angle distribution
- **`reflected`** (*`bool`*) – if the user wishes to activate reflected radiations

Returns

Vegetation types contains clumoing effect ratio, leaf angle distribution and reflectance/transmittance of leaves for each specy

Return type

PyRATP.pyratp.vegetation.Vegetation

`RATPinputs.`**`RATP_meteo`**(*energy*, *day*, *hour*, *coordinates*, *parunit*, *truesolartime*, *direct*, *diffus*)

Initialise a RATP MicroMeteo object from LightVegeManager input parameters

Parameters

- **energy** (*float*) – input ray energy
- **day** (*int*) – input day
- **hour** (*int*) – input hour
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **parunit** (*string*) – unit of energy argument, "micromol.m-2.s-1" or else
- **truesolartime** (*bool*) – activates true solar time or local time to compute sun position
- **direct** (*bool*) – if direct rays are activated
- **diffus** (*bool*) – if diffuse rays are activated

Returns

input meteorological data at current time step

Return type

PyRATP.pyratp.micrometeo.MicroMeteo

`RATPinputs.`**`RdRsH`**(*Rg*, *DOY*, *heureTU*, *latitude*)

fraction diffus/Global en fonction du rapport Global(Sgd)/Extraterrestre(Sod)- pas de temps horaire

2.7.4 VTK module

VTK

Writes VTK files from LightVegeManager geometry and lighting data. Used for visualisation We recommend the software Paraview for visualisation

`VTK.`**`VTKtriangles`**(*trimesh*, *var=[]*, *varname=[]*, *filename=''*)

Writes VTK files from a triangulation mesh. Possibility to associate physical values to the triangles

Parameters

- **trimesh** (*dict of list*) – triangles mesh aggregated by indice elements

```
{ id : [triangle1, triangle2, ...]}
```

- **var** (*list of list*) – list of physical values associated to each triangle. Each element of var is a physical value for each triangle. Then, for n triangles and m values, var looks like:

```
var = [[var1_1, ..., var1_n], ..., [varm_1, ..., varm_n]]
```

- **varname** (*lits of string*) – list of variable names
- **filename** (*string*) – name and path for the output file

`VTK.`**`VTKline`**(*start*, *end*, *filename*)

Writes a VTK file representing a line

Parameters

- **start** (*list*) – starting point `[x, y, z]` of the line

- **end** (*list*) – ending point `[x, y, z]` of the line
- **filename** (*string*) – name and path for the output file

`VTK.`**`ratp_prepareVTK`**(*ratpgrid*, *filename*, *columns=[]*, *df_values=None*)

Sets and prepare data to write a voxel grid in VTK format from a RATP grid

Possibility to sets physical values to each voxel. Otherwise, it will assign leaf area density for each voxel.

Then, it runs :func:VTKvoxels to write a VTK file with the grid of voxels.

Parameters

- **ratpgrid** (*pyratp.grid*) – RAPT grid of voxels
- **filename** (*string*) – name and path for the output file
- **columns** (*list, optional*) – list of columns name of `df_values` to associate to each voxel, defaults to []
- **df_values** (*pandas.Dataframe, optional*) – dataframe with a `"Voxel"` and a `"VegetationType"` columns, matching ratpgrid. It stores physical values associated to each voxel that you want to print, defaults to None

`VTK.`**`VTKvoxels`**(*grid*, *datafields*, *varnames*, *nomfich*)

Writes a VTK file with a grid of voxels and associated physical values.

Display Voxels colored by variable with Paraview

RATP Grid is written in VTK Format as a structured grid

Parameters

- **grid** (*pyratp.grid*) – the RATP grid
- **datafields** (*list of list*) – a list of 3 arrays composed of the a RATP variable to be plotted, corresponding entities, and Voxel ID
  - [0] : kxyz (numpy.array)
  - [1] : entities (numpy.array)
  - [2, …, n] : variable_0, … variable_n-2 (numpy.array)
- **varnames** (*list of string*) – name of the variable to be plotted
- **nomfich** (*string*) – the VTK filename and path

`VTK.`**`PlantGL_to_VTK`**(*scenes*, *path*, *i=0*, *in_unit='m'*, *out_unit='m'*)

Directly converts a list plantGL scenes to a single VTK file The routine aggregates all the scenes in one global triangulation and then calls :func:VTKtriangles Possbility to rescale measure unit scenes

Parameters

- **scenes** (*lits of plantgl.Scene*) – list of scenes to be written
- **path** (*string*) – path of the file. File name is set by default with `"triangles_plantgl_"+str(i)+".vtk"`
- **i** (*int, optional*) – id of the file. Used if you want to batch a large number of scenes, defaults to 0
- **in_unit** (*str, optional*) – input measure unit, defaults to “m”
- **out_unit** (*str, optional*) – output measure unit, defaults to “m”

Raises

`ValueError` – in_unit or out_unit not a valid entry upon the listed measure units

2.7.5 `basicgeometry` module

basicgeometry

Provides basic geometric operations in 3D used by LightVegeManager. Vectors are represented by their cartesian coordinates in a 3-tuple of floats such as v = (x, y, z) Triangles are a list of 3 vectors representing its vertices

`basicgeometry.`**`crossproduct`**`(`*v1*, *v2*`)`

Crossproduct between two vectors v1^v2

Parameters

- **v1** (*3-tuple*) – vector 1
- **v2** (*3-tuple*) – vector 2

Returns

crossproduct

Return type

3-tuple

`basicgeometry.`**`middle`**`(`*v1*, *v2*`)`

Middle point between v1 and v2

Parameters

- **v1** (*3-tuple*) – vector 1
- **v2** (*3-tuple*) – vector 2

Returns

middle point $p = (v1 + v2)/2$

Return type

3-tuple

`basicgeometry.`**`triangle_normal`**`(`*triangle*`)`

Computes normal of an oriented triangle (3D)

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters

triangle (*list*) – triangle represented by its 3 vertices

Returns

normalized vector

Return type

3-tuple

basicgeometry.**triangle_elevation**(*triangle*)

Computes normal elevation of triangle

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters
triangle (*list*) – triangle represented by its 3 vertices

Returns
angle in degree elevation starts from ground to normal vector must be between 0 and 90°

Return type
float

basicgeometry.**triangle_area**(*triangle*)

triangle area

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters
triangle (*list*) – triangle represented by its 3 vertices

Returns
area

Note: algorithm is a copy of `_surf` in `alinea.caribu.caributriangleset`

Return type
float

basicgeometry.**triangle_barycenter**(*triangle*)

triangle barycenter

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters
triangle (*list*) – triangle represented by its 3 vertices

Returns
isobarycenter $(s1 + s2 + s3)/3$

Return type
float

basicgeometry.**rescale**(*triangles*, *h*)

Multiplies all triangle vertices by a ratio h

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters

- **triangles** (*list of list*) – list of triangles in the same format as below
- **h** (*float*) – ratio

Returns
list of triangles with each triangle from input rescaled by h

Return type
list of list

example

```
>>> triangles = [[(0,0,0), (0,1,0), (0,1,1)], [(0,0,1), (0,0,0), (1,0,1)]]
>>> rescale(triangles, 3)
[[(0,0,0), (0,3,0), (0,3,3)], [(0,0,3), (0,0,0), (3,0,3)]]
```

`basicgeometry.`**`translate`**(*triangles*, *tvec*)

Moves a list of triangles with a translation of vector tvec

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters

- **triangles** (*list of list*) – list of triangles in the same format as below
- **tvec** (*3-tuple*) – vector

Returns
list of triangles with each triangle from input translated by tvec

Return type
list of list

example

```
>>> triangles = [[(0,0,0), (0,1,0), (0,1,1)], [(0,0,1), (0,0,0), (1,0,1)]]
>>> tvec = (3,0,1)
>>> translate(triangles, tvec )
[[(3,0,1), (3,1,1), (3,1,2)], [(3,0,2), (3,0,1), (4,0,2)]]
```

`basicgeometry.`**`zrotate`**(*triangles*, *omegadeg*)

Rotates a list of triangles in the xy plane from an angle

Note: a triangle is `[(x1,y1,z1),(x2,y2,z2),(x3,y3,z3)]`

Parameters

- **triangles** (*list of list*) – list of triangles in the same format as below
- **omegadeg** (*float*) – angle in degree

Returns
list of triangles with each triangle from input rotated around z axis by omegadeg

Return type
list of list

example

```
>>> triangles = [[(0,0,0), (0,1,0), (0,1,1)], [(0,0,1), (0,0,0), (1,0,1)]]
>>> zrotate(triangles, 90 )
[[(0,0,0), (-1,0,0), (-1,0,1)], [(0,0,1), (0,0,0), (0,1,1)]]
```

2.7.6 buildRATPscene module

buildRATPscene

Creation of a PyRATP.grid from inputs. The following functions create and initialize a grid from either a triangles mesh, a l-egume grid or an empty geometric input

The argument `parameters` refers to one the three inputs dict of LightVegeManager. It is structured as so:

```
ratp_args = {
        # Grid specifications
        "voxel size" : [dx, dy, dz],
        "voxel size" : "dynamic",

        "origin" : [xorigin, yorigin, zorigin],
        "origin" : [xorigin, yorigin],

        "number voxels" : [nx, ny, nz],
        "grid slicing" : "ground = 0."
        "tesselation level" : int

        # Leaf angle distribution
        "angle distrib algo" : "compute global",
        "angle distrib algo" : "compute voxel",
        "angle distrib algo" : "file",

        "nb angle classes" : int,
        "angle distrib file" : filepath,

        # Vegetation type
        "soil reflectance" : [reflectance_band0, reflectance_band1, ...],
        "reflectance coefficients" : [reflectance_band0, reflectance_band1, ...],
        "mu" : [mu_scene0, mu_scene1, ...]
    }
```

See also:

For more details *Inputs description*

`buildRATPscene.`**`extract_grid_origin`**(*parameters*, *minmax*)
Create grid origin from either input parameters or from the scene

Parameters

- **parameters** (*dict*) – RATP parameters from inputs of LightVegeManager
- **minmax** (*[3-tuple, 3-tuple]*) – list of mininuml point and maximum point in a triangles mesh

Returns

origin of the grid depending of input parameters and size of the mesh

Return type

3-tuple

buildRATPscene.**build_RATPscene_from_trimesh**(*trimesh*, *minmax*, *triLmax*, *matching_ids*, *parameters*, *coordinates*, *reflected*, *infinite*, *stems_id=None*, *nb_input_scenes=0*, *fullgrid=False*)

Build a RATP grid from a triangles mesh.

Parameters

- **trimesh** (*dict*) – triangles mesh aggregated by indice elements .. code-block:: { id : [triangle1, triangle2, …]}
- **minmax** (*[3-tuple, 3-tuple]*) – list of mininuml point and maximum point in a triangles mesh
- **triLmax** (*float*) – longest side of all the triangles in trimesh
- **matching_ids** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice .. code:: matching_ids = { new_element_id : (input_element_id, specy_id)}
- **parameters** (*dict*) – RATP parameters from inputs of LightVegeManager
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **reflected** (*bool*) – if the user wishes to activate reflected radiations
- **infinite** (*bool*) – if the user wishes to activate infinitisation of the grid
- **stems_id** (*list of 2-tuple, optional*) – list of potential stems element in the input scenes, defaults to None
- **nb_input_scenes** (*int, optional*) – number of input scenes in the geometry dict, defaults to 0

Returns

It returns 3 objects:

- **ratpgrid**
  pyratp.grid filled with areas of triangles in trimesh and input parameters
- **matching_tri_vox**
  dict where key is a triangle indice and value the matching voxel indice where the barycenter of the triangle is located
- **distrib**
  dict with a `"global"` key and if `["angle distrib algo"] = "compute voxel"` a second entry where key is `"voxel"`

  value for `"voxel"` is a `numpy.array` of dimension `(numberofvoxels, numberofentities,numberofclasses)`

  value for `"global"` is a list of `numberofentities` list of `numberofclasses` elements

Return type
pyratp.grid, dict, dict

buildRATPscene.**build_RATPscene_empty**(*parameters*, *minmax*, *coordinates*, *infinite*)

Build a RATP grid from an empty geometric input

Parameters

- **parameters** (*dict*) – RATP parameters from inputs of LightVegeManager
- **minmax** (*[3-tuple, 3-tuple]*) – list of mininuml point and maximum point in a triangles mesh
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **infinite** (*bool*) – if the user wishes to activate infinitisation of the grid

Returns

- `ratpgrid`: pyratp.grid sets with input parameters
- `distrib`: dict with one entry `"global" : [1.]`

Return type
pyratp.grid, dict

buildRATPscene.**legumescene_to_RATPscene**(*legumescene*, *parameters*, *coordinates*, *reflected*, *infinite*)

Creates a RATP grid of voxels from a l-egume grid format

Parameters

- **legumescene** (*dict*) – l-egume grid represented by a dict with two entries:
  - `"LA"`: equivalent to m_lais in l-egume, a numpy.array of dimension `(nent, nz, ny, nx)` which represents leaf area in each voxel for each specy
  - `"distrib"`: equivalent to ls_dif in l-egume, a numpy.array of dimension `(nent, nclasses)` which represents the global leaf angle distribution for each input specy

  Note: legumescene is the only input geometric scene which can handle several species

- **parameters** (*dict*) – RATP parameters from inputs of LightVegeManager
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **reflected** (*bool*) – if the user wishes to activate reflected radiations
- **infinite** (*bool*) – if the user wishes to activate infinitisation of the grid

Returns

It returns 3 objects

- `ratpgrid` : pyratp.grid filled with areas of triangles in trimesh and input parameters
- `distrib` : dict with only a `"global"` key, value for `"global"` is a list of `numberofentities` list of `numberofclasses` elements
- `nb0` : number of empty layers between top canopy and maximum layer in legumescene

Return type
pyratp.grid, dict, int

2.7.7 `defaultvalues` module

defaultvalues

Default for simulation fixed parameters These parameters are sets in LightVegeManager if the user did not precise thoses inputs

`defaultvalues.`**`default_LightVegeManager_inputs`**`()`

returns default parameters for LightVegeManager constructor `__init__`

Returns
environment and light model parameters

Return type
dict, dict

2.7.8 `leafangles` module

leafangles

Handles leaf angle distribution, both in its dynamic computing or reading a file

An angle distribution is a list of n elements, where n is the number of angle class between 0 and 90°. Each element of this list is a percentage for leaves to be oriented from 0 to 90/n degree. The sum of all the list entries must equal 1.

`leafangles.`**`read_distrib_file`**`(`*path*, *numberofentities*`)`

Reads global leaf angle distribution in a file the file must matches the format : one specy distribution per line each percentage separated by a coma ','

example

```
0.1382,0.1664,0.1972,0.1925,0.1507,0.0903,0.0425,0.0172,0.005
0.3,0.1,0.15,0.2,0.05,0.2
```

Parameters

- **`path`** (*`string`*) – path to the file
- **`numberofentities`** (*`int`*) – number of species, aka numbre of lines in the file

Returns
distribution for each specy, number of entries on one line

Return type
list of list

`leafangles.`**`compute_distrib_globale`**`(`*trimesh*, *matching_ids*, *numberofclasses*`)`

Calculation of a global leaf angle distribution from a triangle mesh

Parameters

- **`trimesh`** (*`dict`*) – triangles mesh aggregated by indice elements `{ id : [triangle1, triangle2, ...]}`
- **`matching_ids`** (*`dict`*) – dict that matches new element indices in trimesh with specy indice and input element indice, `matching_ids = { new_element_id : (input_element_id, specy_id)}`

  this dict allows us to look how species there is the inputs geometric data

- **numberofclasses** (*int*) – number angle class wanted between 0 and 90 degree

Returns
a leaf angle distribution for each specy. Each distribution is a length numberofclasses the distribution is computed on all trimesh

Return type
list of list

`leafangles.`**`compute_distrib_voxel`**(*trimesh*, *matching_ids*, *numberofclasses*, *numberofvoxels*, *matching_tri_vox*)

Calculation of a local leaf angle distribution from a triangle mesh on each voxel of a grid

Parameters

- **trimesh** (*dict*) – triangles mesh aggregated by indice elements `{ id : [triangle1, triangle2, ...]}`
- **matching_ids** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, `matching_ids = { new_element_id : (input_element_id, specy_id)}` this dict allows us to look how species there is the inputs geometric data
- **numberofclasses** (*int*) – number angle class wanted between 0 and 90 degree
- **numberofclasses** – number of non empty voxels in the grid
- **matching_tri_vox** (*dict*) – dict where key is a triangle indice and value the matching voxel indice where the barycenter of the triangle is located

Returns
array of dimension [number of voxels][number of species][number of angle classes] i.e. it returns for each voxel a leaf angle distribution for each specy

Return type
numpy.array

2.7.9 `outputs` module

outputs

Manages and reformat output results from the light models in pandas.Dataframe with similar columns names.

Light models managed in this module:

- RATP
- CARIBU

Column names for voxels:

- `'VegetationType'`: id of the specy
- `'Iteration'`: RATP can handle multiple iterations in inputs, but it is not properly exploited in LightVegeManager yet
- `'Day'`: input day if LightVegeManager
- `'Hour'`: input hour if LightVegeManager
- `'Voxel'`: index of current voxel
- `'Nx'`: index in grid following x axis, corresponds to pyrapt.grid.numx

- `'Ny'`: index in grid following y axis, corresponds to pyrapt.grid.numy
- `'Nz'`: index in grid following z axis, corresponds to pyrapt.grid.numz
- `'ShadedPAR'`: shaded energy in the current voxel, 0 if no direct light in the input
- `'SunlitPAR'`: sunlit energy
- `'ShadedArea'`: leaf area in voxel which is shaded
- `'SunlitArea'`: leaf area in voxel which is sunlit
- `'Area'`: total leaf area in the voxel
- `'PARa'`: (ShadedPAR * ShadedArea + SunlitPAR * SunlitArea) / (ShadedArea + SunlitArea)
- `'Intercepted'`: relative portion of input rays intercepted by the voxel
- `'Transmitted'`: relative portion of input rays leaving out the voxel

Column names for triangles:

- `'VegetationType'`: id of the specy
- `'Day'`: input day if LightVegeManager
- `'Hour'`: input hour if LightVegeManager
- `'Triangle'`: indice of the triangle
- `'Organ'`: element where the triangle belongs

if ligh model is CARIBU:

- `'Area'`: area of the triangles

for each bandwidth in the inputs you have two entries:

- `band + " Eabs"`: energy absorbed by the triangle
- `band + " Ei"`: energy intercepted by the triangle

If light model is RATP:

- `'Area'`: area of total leaf area in the voxel where the triangle is located
- `'primitive_area'`: area of the triangle
- `'Voxel'`: index of the voxel where the triangle is located
- `'Nx'`: index in grid following x axis, corresponds to pyrapt.grid.numx
- `'Ny'`: index in grid following y axis, corresponds to pyrapt.grid.numy
- `'Nz'`: index in grid following z axis, corresponds to pyrapt.grid.numz
- `'ShadedPAR'`: shaded energy in the current voxel
- `'SunlitPAR'`: sunlit energy, 0 if no direct light in the input
- `'ShadedArea'`: leaf area in voxel which is shaded
- `'SunlitArea'`: leaf area in voxel which is sunlit
- `'PARa'`: (ShadedPAR * ShadedArea + SunlitPAR * SunlitArea) / (ShadedArea + SunlitArea)
- `'Intercepted'`: relative portion of input rays intercepted by the voxel
- `'Transmitted'`: relative portion of input rays leaving out the voxel

Column names for elements:

- `'Day'`: input day if LightVegeManager
- `'Hour'`: input hour if LightVegeManager
- `'Organ'`: id of current element
- `"VegetationType"`: id of the specy
- `"Area"`: area of the element (sum of all triangles belonging to current element)

if light model is CARIBU, for each bandwidth in the optical inputs you have two entries:

- `band + " Eabs"`: energy absorbed by the element, integrated on all triangles belonging to current element
- `band + " Ei"`: energy intercepted by the triangle, integrated on all triangles belonging to current element

if light model is RATP, the following entries are integrated on all triangles belonging to current element with E, current element, t triangles in E and V an entry: $\frac{\sum_{t \in E} area_t * V_t}{area_E}$

- `"PARa"`
- `"Intercepted"`
- `"Transmitted"`
- `"SunlitPAR"`
- `"SunlitArea"`
- `"ShadedPAR"`
- `"ShadedArea"`

`outputs.`**`out_ratp_empty_grid`**`(`*day*, *hour*`)`

Returns an empty dataframe results for RATP

Parameters

- **day** (*int*) – day of simulation
- **hour** (*int*) – hour of simulation

Returns

returns a dataframe with all the keys expected with a RATP simulation but results are empty used if input geometry is empty

Return type

pandas.DataFrame

`outputs.`**`out_ratp_voxels`**`(`*ratpgrid*, *res*, *parunit*`)`

Converts RATP results to pandas dataframe compared to the voxels

Parameters

- **ratpgrid** (*pyratp.grid*) – RATP grid with voxels informations
- **res** (*pyratp.ratp.out_rayt*) – output table of RATP
- **parunit** (*string*) – energy unit of input

  RATP expects W.m-2 in input and return results in µmol.s-1.m-2. if `parunit="W.m-2"` the outputs is converted to W.m-2, otherwise results are in µmol.s-1.m-2

Returns
res.T converted in a pandas Dataframe with voxels relative columns. The soil is not part of the results

Return type
pandas.DataFrame

`outputs.out_ratp_triangles`(*trimesh*, *matching_ele_ent*, *matching_tr_vox*, *voxels_outputs*)

Converts RATP results to pandas dataframe compared to the triangles (and voxels)

Parameters

- **trimesh** (*dict*) – triangles mesh aggregated by indice elements .. code-block:: { id : [triangle1, triangle2, …]}
- **matching_ele_ent** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, .. code:: matching_ids = { new_element_id : (input_element_id, specy_id)}
- **matching_tr_vox** (*dict*) – dict where key is a triangle indice and value the matching voxel indice where the barycenter of the triangle is located
- **voxels_outputs** (*pandas.Dataframe*) – output of :func:out_ratp_voxels

Returns
output of :func:out_ratp_voxels merged with associated triangles

Return type
pandas.Dataframe

`outputs.out_ratp_elements`(*matching_ele_ent*, *reflected*, *reflectance_coef*, *trianglesoutputs*)

If trimesh is not empty, aggregates RATP results by element (keys in dict trimesh)

Parameters

- **matching_ele_ent** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, .. code:: matching_ids = { new_element_id : (input_element_id, specy_id)}
- **reflected** (*bool*) – if the user wishes to activate reflected radiations
- **reflectance_coef** (*list of list*) – coefficient for each specy and each input bandwidth
- **trianglesoutputs** (*pandas.Dataframe*) – output of :func:out_ratp_triangles

Returns
dataframe integrated on element informations

Return type
pandas.Dataframe

`outputs.out_caribu_mix`(*rdrs*, *c_scene_sun*, *c_scene_sky*, *raw_sun*, *aggregated_sun*, *raw_sky*, *aggregated_sky*, *issensors*, *issoilmesh*)

Mix diffuse and direct rays results according to a ratio rdrs

Parameters

- **rdrs** (*float*) – Spitters's model estimating for the diffuse:direct ratio
- **c_scene_sun** (*CaribuScene*) – CaribuScene with direct lighting computed (from a sun)
- **c_scene_sky** (*CaribuScene*) – CaribuScene with diffuse lighting computed (from a sky)
- **raw_sun** (*dict of dict*) – results of c_scene_sun for triangles

- **aggregated_sun** (*dict of dict*) – results of c_scene_sun aggregated by element
- **raw_sky** (*dict of dict*) – results of c_scene_sky for triangles
- **aggregated_sky** (*dict of dict*) – results of c_scene_sky aggregated by element
- **issensors** (*bool*) – if option virtual sensors is activated
- **issoilmesh** (*bool*) – if option soil mesh is activated

Returns

if sensors is activated, it extracts its results from aggregated_sun and aggregated_sky

if soilmesh is activated, it extracts its results from c_scene_sun and c_scene_sky

then it mixes raw, aggregated, virtual sensors and soil energy results such as

`result = rdrs * diffuse_result + (1 - rdrs) * direct_result`

Return type

dict of dict, dict of dict, dict of list, dict of float

`outputs.`**`out_caribu_nomix`**(*c_scene*, *aggregated*, *issensors*, *issoilmesh*)

Extracts only sensors and soilmesh results if activated

Parameters

- **c_scene** (*CaribuScene*) – instance of CaribuScene containing geometry, light source(s), opt etc…
- **aggregated** (*dict of dict*) – result of CaribuScene.run
- **issensors** (*bool*) – if option virtual sensors is activated
- **issoilmesh** (*bool*) – if option soil mesh is activated

Returns

extracts sensors results from aggregated and soil energy from c_scene

Return type

dict, dict

`outputs.`**`out_caribu_elements`**(*day*, *hour*, *trimesh*, *matching_ids*, *aggregated*, *sun_up*, *caribu_triangles={}*)

Converts aggregated in a pandas.Dataframe following indices in LightVegeManager

Parameters

- **day** (*int*) – day of simulation
- **hour** (*int*) – hour of simulation
- **trimesh** (*dict*) – triangles mesh aggregated by indice elements .. code-block:: { id : [triangle1, triangle2, …]}
- **matching_ele_ent** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, .. code:: matching_ids = { new_element_id : (input_element_id, specy_id)}
- **aggregated** (*dict of dict*) – result of CaribuScene.run
- **sun_up** (*bool*) – if sun elevation is > 2° and direct rays are > 0 W.m^2

Returns

aggregated results rearrange in a Dataframe with element correspondance in LightVegeManager

Return type
pandas.Dataframe

`outputs.`**`out_caribu_triangles`**`(`*day*, *hour*, *trimesh*, *matching_ids*, *raw*, *sun_up*`)`

Converts raw in a pandas.Dataframe following simulation datas

Parameters

- **day** (*int*) – day of simulation
- **hour** (*int*) – hour of simulation
- **trimesh** (*dict*) – triangles mesh aggregated by indice elements .. code-block:: { id : [triangle1, triangle2, . . .]}
- **matching_ele_ent** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice, .. code:: matching_ids = { new_element_id : (input_element_id, specy_id)}
- **raw** (*dict of dict*) – triangles results from CaribuScene.run
- **sun_up** (*bool*) – if sun elevation is > 2° and direct rays are > 0 W.m^2

Returns
triangle results rearrange in a Dataframe

Return type
pandas.Dataframe

2.7.10 `plantGL` module

plantGL

visualisation plantGL

`plantGL.`**`cscene_to_plantGLScene_stems`**`(`*cscene*, *stems_id=None*, *matching_ids={}*`)`

Transform a triangles mesh to a plantGL scene, whith a color difference between leaves and stems

Parameters

- **cscene** (*dict of list*) – LightVegeManager triangulations mesh
- **stems_id** (*list of tuple, optional*) – list of tuple (element_id, specy_id), defaults to None
- **matching_ids** (*dict of tuple, optional*) – dict that matches new element indices in trimesh with specy indice and input element indice, defaults to {} `matching_ids = { new_element_id : (input_element_id, specy_id)}`

Returns
cscene in plantGL scene format with leaves in green and stems in brown

Return type
plantGL Scene

`plantGL.`**`cscene_to_plantGLScene_light`**`(`*cscene*, *outputs={}*, *column_name='par Ei'*`)`

Transform a triangles mesh to a plantGL scene, colors represent lightint results from blue to red

Parameters

- **cscene** (*dict of list*) – LightVegeManager triangulations mesh

- **outputs** (*Pandas.Dataframe, optional*) – outputs results from LightVegeManager, defaults to { }
- **column_name** (*str, optional*) – name of the value to plot, defaults to "par Ei"

Returns
cscene in plantGL scene format with a color from blue to red

Return type
plantGL Scene

`plantGL.ratpgrid_to_plantGLScene`(*ratpgrid*, *transparency=0.0*, *plt_cmap='Greens'*, *outputs={}*)

Transform a RATP grid mesh to a plantGL scene Colors can either follows leaf area or lighting values

Parameters

- **ratpgrid** (*pyratp.grid*) – grid of voxels in RATP format
- **transparency** (*float, optional*) – transparency value of the voxels from 0 to 1, defaults to 0.
- **plt_cmap** (*str, optional*) – name of the colormap to color the voxels, defaults to "Greens"
- **outputs** (*pandas.Dataframe, optional*) – lighting results from LightVegeManager, defaults to { }

Returns
ratpgrid in plantGL scene

Return type
plantGL Scene

2.7.11 sky module

sky

Build a sky from LightVegeManager input informations

Possible sky options:

- `"turtle46"` : soc sky composed by 46 directions in a turtle shape
- filepath : string indicating a file path
- `[nb_azimut, nb_zenith, "soc" or "uoc"]` : build a sky with a custom number of directions

output :

- with RATP : pyratp.skyvault
- with CARIBU : [(weight, (dir[0], dir[1], dir[2])), …]

Sky file is in RATP format

`sky.ratpformat_to_caribuformat`(*az*, *h*, *pc*, *rad=True*)

Converts sky informations from RATP format to CARIBU format

Parameters

- **az** (*list*) – azimuts of each sky direction (degree)
- **h** (*list*) – elevation of each sky direction (degree)

- **pc** (*list*) – perez weight of each sky direction

Returns
direction vectors and weights for each sky direction list of (weight, (x, y, z)) vector is from sky to ground

Return type
list of tuple

`sky.caribuformat_to_ratpformat(directions)`

Converts CARIBU format to RATP sky format

Parameters
directions (*list of tuples*) – list of all each sky directions, such as one direction is (perez coeff, (x, y, z)) where vector looks from sky to floor

Returns

- pc : perez coefficients for each direction
- az : azimut of each direction, x+ = 0.
- h : elevation of each direction

:rtype:list, list, list

`sky.RATPsky(skytype)`

Creates a pyratp.skyvault

Parameters
skytype (*string or list*) – the environment["sky"] from the LightVegeManager inputs. It is either: * `"turtle46"` * a filepath * [nb_azimut, nb_elevation, "soc" or "uoc"], nb_azimut is the number of azimut directions and nb_elevation the number of zenital directions for cutting out the sky

Raises
ValueError – if skytype is not one of the curretn 3 possibilities

Returns
a sky in RATP format

Return type
pyratp.skyvault

`sky.CARIBUsky(skytype)`

Build a sky in CARIBU format

Parameters
skytype (*string or list*) – the environment["sky"] from the LightVegeManager inputs. It is either:

- `"turtle46"`
- a filepath
- [nb_azimut, nb_elevation, "soc" or "uoc"], nb_azimut is the number of azimut directions and nb_zenith the number of zenital directions for cutting out the sky

Raises
ValueError – if skytype is not one of the curretn 3 possibilities

Returns
a list of the directions representing the sky. each entry of the list is a tuple (weight, vector), where

weight is a float for the weight the direction and vector, a tuple (x, y, z), representing the position of the sky direction, from sky to the ground

Return type
list of tuple

`sky.discrete_sky(n_azimuts, n_zeniths, sky_type)`

Cuts out a sky following input number of directions

Parameters

- **n_azimuts** (*int*) – number of azimut directions of the sky
- **n_zeniths** (*int*) – number of elevations directions of the sky
- **sky_type** (*string*) – `"soc"` or `"uoc"`

Returns
4 output lists of length n_azimuts * n_zeniths * ele: elevations in degrees (angle soil to sky) * azi: azimuts in degrees * omega: solid angle of directions * pc: relative contribution of directions to incident diffuse radiation

Return type
list, list, list, list

2.7.12 `stems` module

stems

Manages stems element (opaque)

`stems.extract_stems_from_MTG(MTG, entity_id)`

Extracts stems element from a MTG table

Parameters

- **MTG** (*openalea.mtg.mtg.MTG*) – MTG table descripting plant elements
- **entity_id** (*int*) – indice of the corresponding geometric scene of the MTG table scene in the LightVegeManager inputs

Returns
`stems` is a list where each entry is a tuple (element indice, entity_id)

Return type
list of tuple

`stems.manage_stems_for_ratp(stems_id, matching_ids, ratp_parameters)`

Adds a specy to separate stems from leaves in vegetation parameters

Parameters

- **stems_id** (*list of tuple*) – list of tuple (element_id, specy_id)
- **matching_ids** (*dict*) – dict that matches new element indices in trimesh with specy indice and input element indice `matching_ids = { new_element_id : (input_element_id, specy_id)}`
- **ratp_parameters** (*dict*) – RATP parameters from inputs of LightVegeManager

Raises

`ValueError` – if too many stems elements are identified comparing to the total number of elements cumulated over all species

2.7.13 `sun` module

sun

Build a sun respecting each light model format

For computing the sun position you can either use CARIBU or RATP algorithm, which slightly change in the process

`sun.ratp_sun`(*day*, *hour*, *coordinates*, *truesolartime*)

Converts output RATP sundirection routine to CARIBU light format

Parameters

- **day** (*int*) – input day
- **hour** (*int*) – input hour
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **truesolartime** (*bool*) – activates true solar time or local time to compute sun position

Returns

sun direction in a tuple with cartesian coordinates (x,y,z), vector is oriented from sky to ground

Return type

tuple

`sun.caribu_sun`(*day*, *hour*, *coordinates*, *truesolartime*)

Compute sun direction from CARIBU algorithm

Parameters

- **day** (*int*) – input day
- **hour** (*int*) – input hour
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **truesolartime** (*bool*) – activates true solar time or local time to compute sun position

Returns

sun direction in a tuple with cartesian coordinates (x,y,z), vector is oriented from sky to ground

Return type

tuple

`sun.print_sun`(*day*, *hour*, *coordinates*, *truesolartime*)

Prints sun position ouputs from RATP and CARIBU algorithm with the same inputs

Parameters

- **day** (*int*) – input day
- **hour** (*int*) – input hour
- **coordinates** (*list*) – [latitude, longitude, timezone]
- **truesolartime** (*bool*) – activates true solar time or local time to compute sun position

2.7.14 `tesselator` module

tesselator

Handles tesselation of a triangulation, the subdivision of triangulation. The tesselation was initially made to have a better matching of a triangle mesh in a grid of voxels and avoid triangle to be located in several voxels

Tesselation operates as a recursive function until a certain level is reached

`tesselator.`**`whichvoxel`**`(`*p*, *mygrid*`)`

Returns in which voxel p is located

Parameters

- **p** (*list or tuple*) – 3D vector located in the grid
- **mygrid** (*pyratp.grid*) – RATP grid

Returns

returns the voxels where p is located in a list [kx, ky, kz] where ki is the indice voxel on each axis, [0, 0, 0] is origin voxel

Return type

list

`tesselator.`**`samevoxel`**`(`*voxels*`)`

Check if all voxels in list voxels are the same

Parameters

voxels (*list*) – list of voxels in format [kx, ky, kz]

Returns

if all elements of voxels are the same

Return type

bool

`tesselator.`**`tesselate`**`(`*mygrid*, *triangle*`)`

Check if the triangle is strictly inside a voxel or between several voxels. If so, it cuts out triangle in 4 smaller triangles.

Parameters

- **mygrid** (*pyratp.grid*) – RATP voxels grid
- **triangle** (*list of tuple*) – triangle represented by its 3 vertices `triangle = [(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)]`

Returns

- if the triangle is inside a voxel, it returns the input triangle in a 1-list
- else if the triangle is between several voxels, it cuts out triangle in four smaller triangles from its vertices and barycenter

Return type

list of triangle

`tesselator.`**`tesselate2`**`(`*triangle*`)`

Same as :func:tesselate but without the grid matching

Parameters
`triangle` (*list of tuple*) – triangle represented by its 3 vertices `triangle = [(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)]`

Returns
it cuts out triangle in four smaller triangles from its vertices and barycenter

Return type
list of triangle

tesselator.**iterate_trianglesingrid**(*triangle*, *mygrid*, *level*, *levelmax*, *triangles_shape*)

Recursion on a triangle, while its subdivision doesn't match an RATP grid or exceed a certain level of tesselation

Parameters

- **triangle** (*list of tuple*) – triangle represented by its 3 vertices `triangle = [(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)]`
- **mygrid** (*pyratp.grid*) – RATP voxels grid
- **level** (*int*) – current level of tesselation, i.e. how many times we cut out the initial triangle
- **levelmax** (*int*) – final level of tesselation to not be exceeded
- **triangles_shape** (*list of triangles*) – list of triangles used to stock new triangles or input `triangle` if no tesselation is computed

Returns
`triangles_shape` updates with input `triangle` or new triangles if tesselation was activated

Return type
list of triangles

tesselator.**iterate_triangles**(*triangle*, *level*, *levelmax*, *triangles_shape*)

Subdivide a triangle until its tesselation reaches levelmax. It stocks the new triangles in `triangle_shape` Recursive function

Parameters

- **triangle** (*list of tuple*) – triangle represented by its 3 vertices `triangle = [(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)]`
- **level** (*int*) – current level of tesselation, i.e. how many times we cut out the initial triangle
- **levelmax** (*int*) – final level of tesselation to not be exceeded
- **triangles_shape** (*list of triangles*) – list of triangles used to stock new triangles or input `triangle` if no tesselation is computed

Returns
`triangles_shape` updates with new triangles from tesselation

Return type
list of triangles

2.7.15 `transfer` module

transfer

Handles transfer of LightVegeManager results to plant Models.

Currently, it manages CN-Wheat and l-egume. Only l-egume needs additionnal processes to converts Dataframe results in a compatible format.

l-egume expects the absorb PAR either per plant or locally following a grid of voxels, and the transmitted PAR locally following a grid of voxels. Here, we will convert and transform results for CARIBU and RATP to those l-egume format.

transfer.**transfer_ratp_legume**(*m_lais*, *energy*, *ratp_grid*, *voxels_outputs*, *nb0*, *epsilon=1e-08*)

Transfers LightVegeManager outputs from RATP to l-egume Absorb and transmitted PAR will follow a RATP grid of voxels, matching the dimensions of intern l-egume grid of voxels.

Parameters

- **m_lais** (*numpy.array*) – leaf area represented in a numpy.array of dimension [number of species, number of z layers, number of y layers, number of x layers]
- **energy** (*float*) – input energy
- **ratp_grid** (*pyratp.grid*) – RATP grid of voxels
- **voxels_outputs** (*pandas.Dataframe*) – results from LightVegeManager
- **nb0** (*int*) – number of empty layers from top of the canopy and maximum z layers in m_lais
- **epsilon** (*float*) – criteria of minimum intercepted portion of PAR in a non empty voxel

Returns

two array with lighting informations

- `res_abs_i`: absorb PAR in each voxel for RATP grid of voxels. It has the same dimensions as `m_lais`
- `res_trans`: transmitted PAR in each voxel, i.e. the energy leaving the voxels from input rays. This value is not dependent on specy

dimensions are (number of z layers, number of y layers, number of x layers)

Return type

numpy.array, numpy.array

transfer.**transfer_caribu_legume**(*energy*, *skylayer*, *id*, *elements_outputs*, *sensors_outputs*, *sensors_dxyz*, *sensors_nxyz*, *m_lais*, *list_invar*, *list_lstring*, *list_dicFeuilBilanR*, *infinite*, *epsilon*)

Transfers LightVegeManager outputs from CARIBU to l-egume We will update list_invar which stores the total intercepted energy for each plant, and return an array storing transmitted energy following the intern grid of voxels in l-egume. To do so, we used virtual sensors in CARIBU to get incoming radiations in selected locations.

Parameters

- **energy** (*float*) – input energy
- **skylayer** (*int*) – number of empty layers from top of the canopy and maximum z layers in m_lais
- **id** (*list of int*) – list of indices of input scenes associated with l-egume

- **elements_outputs** (*pandas.Dataframe*) – Dataframe results of elements formatted by :func:outputs.out_caribu_elements
- **sensors_outputs** (*dict of list*) – lighting results of virtual sensors form CARIBU in the format for each bandwidth computed,

```
sensors_outputs[band+" Eabs"] = {sensor_id : energy}
sensors_outputs[band+" Ei"] = {sensor_id : energy}
```

- **sensors_dxyz** (*list*) – size of sides of a voxel in the grid of virtual sensors [dx, dy, dz]
- **sensors_nxyz** (*list*) – number of sensors in each direction in the grid [nx, ny, nz]
- **m_lais** (*numpy.array*) – leaf area represented in a numpy.array of dimension [number of species, number of z layers, number of y layers, number of x layers]
- **list_invar** (*list of dict*) – from l-egume, each element corresponds to an input specy of l-egume. Each element is a dict invar stores instant intern variables of l-egume.
- **list_lstring** (*list of dict*) – from l-egume, each element corresponds to an input specy of l-egume. Each element is a dict lstring stores the l-system of each plant
- **list_dicFeuilBilanR** (*list of dict*) – from l-egume, each element corresponds to an input specy of l-egume. Each element is a dict dicFeuiBilanR stores correspondances between voxels grid and each plant
- **infinite** (*bool*) – if the user wishes to activate infinitisation of the grid
- **epsilon** (*float*) – criteria of minimum intercepted portion of PAR in a voxel (if res_abs_i is zero, l-egume will crash)

Raises

ValueError – Virtual sensors and finite scene doesn't work yet with CARIBU

Returns

- it updates `list_invar` and its key entries `"parap"` and `"parip"`, each element if the scipy.array is the sum of all intercepted energy for each plant. This process is a rewrite of `calc_paraF` in ShootMorpho.py module of l-egume, adapted to LightVegeManager numerotation of triangles
- `res_trans` an array of transmitted energy for each voxel in a grid of dimensions `sensors_dxyz * sensors_nxyz`

Return type

numpy.array

2.7.16 `trianglesmesh` module

trianglesmesh

It builds and handles triangulation mesh.

The main triangulation format in LightVegeManager is the CARIBU format: scene (dict): a *{primitive_id: [triangle,]}* dict. A triangle is a list of 3-tuples points coordinates. example:

```
scene = {   # element 0
            0 : [
                    [(1,1,1), (0,0,0), (1,2,2)],  # triangle 0 of element 0
```

(continues on next page)

(continued from previous page)

```
                [(10,10,10), (0,0,0), (10,20,20)] # triangle 1 of element 0
            ] ,

        # element 1
        1 : [
                [(12,12,12), (10,20,30), (21,22,22)],
                [(10,10,10), (10,20,30), (20,20,20)]
            ]
    }
```

See also:

For more details *Inputs description*

`trianglesmesh.`**`triangles_entity`**(*cscene*, *entity_id*, *matching_ids*)

Return a list of triangles belonging to the specy `entity_id`

Parameters

- **cscene** (*dict of list*) – LightVegeManager triangulations mesh
- **entity_id** (*int*) – specy indice in matching_ids
- **matching_ids** (*dict*) – dict that matches new element indices in cscene with specy indice and input element indice, .. code:: matching_ids = { new_element_id : (input_element_id, specy_id)}

  this dict allows us to look how species there is the inputs geometric data

Returns

list of triangles belonging to `entity_id` in `cscene`, a triangle is `[(x1, y1, z1), (x2, y2, z2), (x3, y3, z3)]`

Return type

list

`trianglesmesh.`**`globalid_to_elementid`**(*cscene*, *triangleid*)

Return the element index in cscene from a global triangle indice, as if triangles were in a list without a sorting by element.

Parameters

- **cscene** (*dict of list*) – LightVegeManager triangulations mesh
- **triangleid** (*int*) – indice from 0 to total number of triangles in cscene

Raises

IndexError – if triangleid > total number of triangles in cscene

Returns

element indice where triangleid belongs in cscene

Return type

int

`trianglesmesh.`**`globalid_to_triangle`**(*cscene*, *triangleid*)

Return the corresponding triangle in cscene from a global triangle indice, as if triangles were in a list without a sorting by element.

Parameters

- **cscene** (*dict of list*) – LightVegeManager triangulations mesh
- **triangleid** (*int*) – indice from 0 to total number of triangles in cscene

Raises
`IndexError` – if triangleid > total number of triangles in cscene

Returns
Corresponding triangle in cscene

Return type
list of tuple

`trianglesmesh.`**compute_area_max**(*cscene*)

Maximum triangle area in a triangulation

Parameters
cscene (*dict of list*) – LightVegeManager triangulations mesh

Returns
maximum triangle area in `cscene`

Return type
float

`trianglesmesh.`**compute_minmax_coord**(*cscene*)

Maximum and minimum point in all triangulation mesh

Parameters
cscene (*dict of list*) – LightVegeManager triangulations mesh

Returns
minimum point, maximum point

Return type
list, list

`trianglesmesh.`**compute_trilenght_max**(*cscene*)

Search for maximum side of an axis oriented voxel where all triangles could fit in

Parameters
cscene (*dict of list*) – LightVegeManager triangulations mesh

Returns
maximum side of an axis oriented voxel where all triangles in cscene could fit in

Return type
float

`trianglesmesh.`**isatriangle**(*t*)

`trianglesmesh.`**chain_triangulations**(*scenes*)

Aggregates all input geometric scenes in one global triangulation

Current known formats:

- plantGL scene
- VGX file
- CARIBU triangulation format (which is used in LightVegeManager for the global scene)
- MTG table with `"geometry"` identifier

- l-egume grid: dict of two entries, leaf area through a voxels grid and leaf angle distribution, for each specy

Only l-egume grid can stores multiple species, otherwise each entry scene must represent one specy

Parameters
`scenes` (*list*) – `geometric["scenes"]` from LightVegeManager inputs, list of geometric scenes. Scenes can be in different format in the list.

Returns

it returns 4 objects

- `complete_trimesh`: global triangulation which aggregates all input scenes. It is in CARIBU format
- `matching_ids`: dict which stores correspondances between new element indice, element indice in the input and specy indice
- `legume_grid`: boolean specifing if at least one l-egume grid is among the input scenes
- `id_legume_scene`: indice in the input scenes of a l-egume grid

Return type
dict of list, dict of list, bool, int

`trianglesmesh.`**`vgx_to_caribu`**(*file*, *id_element*)

Reads VGX file and converts them in CARIBU scene format line is a leaf triangle if R column != 42 :param file: path name of the file :type file: string :param id_element: element indice where the triangulation will be stored :type id_element: int :return: triangulation in CARIBU format `{id_element : [triangle,]}` :rtype: dict of list

`trianglesmesh.`**`apply_transformations`**(*cscene*, *matching_ids*, *transformations*, *cscene_unit*)

Applies geometric transformations on some part of the triangulation set in input datas Each transformation applies on all triangles from the same input scene.

Parameters

- **`cscene`** (*`dict of list`*) – LightVegeManager triangulations mesh
- **`matching_ids`** (*`dict`*) – dict that matches new element indices in cscene with specy indice and input element indice,

  ```
  matching_ids = { new_element_id : (input_element_id, specy_id)}
  ```

  this dict allows us to look how species there is the inputs geometric data
- **`transformations`** – dict containing which geometric to apply on which element

  Possible transformations:

  – `["scenes unit] = {id_scene: unit}`

  – `["rescale"] = {id_scene: h}`

  – `["translate"] = {id_scene: vector}`

  – `["xyz orientation"] = {id_scene: orientation}`

`id_scene` is the index in the input geometric scenes list

Parameters
`cscene_unit` (*`string`*) – measure unit of cscene

Raises

- **ValueError** – unit is not in the list `['mm','cm','dm','m','dam','hm','km']`
- **ValueError** – unit is not in the list `['mm','cm','dm','m','dam','hm','km']`
- **ValueError** – xyz orientation is not one of `"x+ = S", "x+ = E", "x+ = W", "x+ = N"`

Returns
`cscene` with transformations applied to inputs scene parts in the global mesh

Return type
dict of list

`trianglesmesh.`**`create_heterogeneous_canopy`**(*geometrical_model*, *mtg=None*, *nplants=50*, *var_plant_position=0.03*, *var_leaf_inclination=0.157*, *var_leaf_azimut=1.57*, *var_stem_azimut=0.157*, *plant_density=250*, *inter_row=0.15*, *id_type=None*, *seed=None*)

Duplicate a plant in order to obtain a heterogeneous canopy.

Parameters

- **nplants** (*int*) – the desired number of duplicated plants
- **var_plant_position** (*float*) – variability for plant position (m)
- **var_leaf_inclination** (*float*) – variability for leaf inclination (rad)
- **var_leaf_azimut** (*float*) – variability for leaf azimut (rad)
- **var_stem_azimut** (*float*) – variability for stem azimut (rad)
- **id_type** (*string*) – precise how to set the shape id of the elements : None, plant or organ

Returns
duplicated heterogenous scene and its domain

Return type
openalea.plantgl.all.Scene, (float)

`trianglesmesh.`**`random_triangle_generator`**(*worldsize=(0, 100)*, *spheresize=(1.0, 1.0)*, *sigma_angle=(3.141592653589793, 3.141592653589793)*, *theta_angle=(0.7853981633974483, 0.6283185307179586)*)

Generate a random based on parameters

Vertices are generated on a surface of a sphere

Args:
worldsize (tuple, optional): min and max where sphere center can be generated. Defaults to (0,100). spheresize (tuple, optional): mean and std of the sphere size. Defaults to (1.,1.). sigma_angle (tuple, optional): mean and std of the spherical angle on xy plane. Defaults to (math.pi, math.pi). theta_angle (tuple, optional): mean and std of the zenithal angle. Defaults to (math.pi/4, math.pi/5).

Returns:
list of 3 3-tuples: triangles defined by 3 xyz points

2.7.17 `voxelsmesh` module

voxelsmesh

Builds and handles axis oriented voxels mesh

`voxelsmesh.`**`compute_grid_size_from_trimesh`**(*pmin*, *pmax*, *dv*, *grid_slicing=None*)

Dynamically compute number of voxels for each axis in the grid The grid is ajusted to be the smallest box containing another mesh

Parameters

- **pmin** (*list*) – Minimum point of a mesh `[x, y, z]`
- **pmax** (*list*) – Maximum point of a mesh `[x, y, z]`
- **dv** (*list*) – size a voxel in each direction `[dx, dy, dz]`
- **grid_slicing** (*string, optional*) – possibility to force the ground to be at z=0, defaults to None

Returns

number of voxels in each direction x, y and z

Return type

int, int, int

`voxelsmesh.`**`tesselate_trimesh_on_grid`**(*trimesh*, *ratpgrid*, *levelmax*)

Loop on all triangles of a triangulation to tesselate them for better matching a grid of voxels Triangles will subdivide on sides of voxels respecting a certain maximum level

Parameters

- **trimesh** (*dict of list*) – triangles mesh aggregated by indice elements

```
{ id : [triangle1, triangle2, ...]}
```

- **ratpgrid** (*pyratp.grid*) – RATP grid of voxels
- **levelmax** (*int*) – maximum level for subdividing triangles

Returns

a copy of trimesh with subdivided triangles

Return type

dict of list

`voxelsmesh.`**`fill_ratpgrid_from_trimesh`**(*trimesh*, *matching_ids*, *ratpgrid*, *stems_id=None*, *nb_input_scenes=0*)

Fills a RATP grid from a triangulation. It gives barycenters and areas of triangles to update the leaf area density in the corresponding voxel.

Parameters

- **trimesh** (*dict of list*) – triangles mesh aggregated by indice elements

```
{ id : [triangle1, triangle2, ...]}
```

- **matching_ids** (*dict*) – dict that matches new element indices in cscene with specy indice and input element indice,

```
matching_ids = { new_element_id : (input_element_id, specy_id)}
```

this dict allows us to look how species there is the inputs geometric data

- **ratpgrid** (*pyratp.grid*) – RATP grid of voxels
- **stems_id** (*list of 2-tuple, optional*) – list of potential stems element in the input scenes, defaults to None
- **nb_input_scenes** (*int, optional*) – number of input geometrical scenes. It can be different with number of species if there is a l-egume grid in the input with several species in it, defaults to 0

Returns
copy of `ratpgrid` with leaf area density values from barycenters and areas of the input triangles

Return type
pyratp.grid

`voxelsmesh.`**`fill_ratpgrid_from_legumescene`**(*legumescene*, *ratpgrid*, *nb0*)

Fills a RATP grid from a l-egume grid It updates `ratpgrid`.

Parameters

- **legumescene** (*dict*) – l-egume grid represented by a dict with two entries:
  - `"LA"`: equivalent to m_lais in l-egume, a numpy.array of dimension `(nent, nz, ny, nx)` which represents leaf area in each voxel for each specy. nz=0 is the top layer
  - `"distrib"`: equivalent to ls_dif in l-egume, a numpy.array of dimension `(nent, nclasses)` which represents the global leaf angle distribution for each input specy

  Note: legumescene is the only input geometric scene which can handle several species

- **ratpgrid** (*pyratp.grid*) – RATP grid of voxels. nz=0 is the top layer
- **nb0** (*int*) – number of empty layers from top of the canopy and maximum z layers in m_lais

`voxelsmesh.`**`reduce_layers_from_trimesh`**(*trimesh*, *pmax*, *dxyz*, *nxyz*, *matching_ids*, *ids=None*)

Number of empty layers in a grid of voxels, from the top of the canopy to the last expected layer. It computes this number from a triangles mesh

Parameters

- **trimesh** (*dict of list*) – triangles mesh aggregated by indice elements

```
{ id : [triangle1, triangle2, ...]}
```

- **pmax** (*list*) – Maximum point of a mesh `[x, y, z]`
- **dxyz** (*list*) – size of sides of a voxel `[dx, dy, dz]`
- **nxyz** (*list*) – number of voxels in each direction `[nx, ny, nz]`, nz is the expected number of layers
- **matching_ids** (*dict*) – dict that matches new element indices in cscene with specy indice and input element indice,

```
matching_ids = { new_element_id : (input_element_id, specy_id)}
```

this dict allows us to look how species there is the inputs geometric data

- **ids** (*list of int, optional*) – list of specy indices considered not empty, defaults to None

Returns
number of empty layers between top of the canopy (represented by `trimesh`) and `nxyz[2]`

Return type
int

CHAPTER THREE

CONTACT

For any question, please submit to https://github.com/mwoussen/lightvegemanager/issues.

CHAPTER

FOUR

AUTHORS

- Maurane Woussen
- Romain Barillot
- Didier Combes
- Gaëtan Louarn

CHAPTER

FIVE

LICENSE

lightvegemanager is released under CeCILL-C License. See file LICENSE for details.

CHAPTER

SIX

FUNDING

CHAPTER

SEVEN

RESEARCH REFERENCES

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX