# Documentation

## Installation de l'outil

1. Création d'un environnement conda avec miniconda3

```
conda create -n myenvname python=3.7 \
                openalea.mtg=2.0.5 openalea.plantgl=3.14.1
openalea.lpy=3.9.2 alinea.caribu=8.0.7 alinea.astk=2.1.0
xlrd=2.0.1\
                 coverage=6.3 nose=1.3.7 sphinx=4.4.0
statsmodels=0.13.1 numpy=1.20.3 scipy=1.7.3 pandas=1.3.4
progressbar2=3.37.1\
                  -c conda-forge -c fredboudon
```

2. Se placer dans l'environnement créé : `conda activate myenvname`
3. Installation de Adel-Wheat (requis par WheatFspm)
    - Linux

```
wget
https://github.com/rbarillot/adel/archive/python3.zi
p
unzip python3.zip && cd adel-python3
python setup.py develop
```

    - Windows
        - télécharger et déziper l'archive : https://github.com/rbarillot/adel/archive/python3.zip
        - lancer la commande suivante dans le dossier `adel-python3`

```
python setup.py develop
```

4. Installation de Wheat-fspm

a. dans une console Git :

```
git clone --recurse-submodules
https://github.com/openalea-incubator/WheatFspm.git
cd WheatFspm
git submodule update --remote
```

b. installation dans l'environnement conda (dans le dossier `WheatFspm`)

```
python setup.py develop
```

5. Installation de l-egume

a. dans une console Git :

```
git clone -b Develop https://github.com/glouarn/l-
egume.git
```

b. installation dans l'environnement conda (dans le dossier `l-egume`)

```
python setup.py develop
```

6. installation de LightVegeManager

a. dans une console Git :

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

b. installation dans l'environnement conda (dans le dossier `lightvegemanager`)

```
python setup.py develop
```

7. Installation de PyRATP en local, compilation avec numpy
   Dans le dossier `lightvegemanager/PyRATP/f90`

- Linux

```
bash compile_pyratp_lin64.sh
```

- Windows

```
.\compile_pyratp_win64.bat
```

/!\ attention : les chemins relatifs de la ligne 5 (appel de `gfortran.exe`) peuvent être erronés, il faudra les modifier manuellement.

## Dictionnaires d'entrée

L'outil demande trois dictionnaires Python en entrée pour la gestion des calculs :

- `environment` : regroupe les paramètres du ciel, les rayonnements à activer (direct, diffus, réfléchis), les paramètres de réflectance, transmittance et reproduction infinie de la scène. Ces paramètres ne changent pas au cours de la simulation.
- `lightmodel_parameters` : regroupe les paramètres propres au programme de calcul de la lumière choisie (dimensions de la grille pour RATP, présence de capteurs dans CARIBU, etc...). Ces paramètres ne changent pas au cours de la simulation.
- `geometry` : regroupe les scènes géométriques, des opérations à leurs appliquer et certaines descriptions pertinentes dans la suite des calculs.

Seules les scènes géométriques, contenues dans le dictionnaire `geometry` sont susceptibles d'évoluer au cours de la simulation contrairement aux deux autres qui regroupent uniquement des paramètres fixes.

### Description de `environment`

#### Clés d'entrée

- `"coordonates"` : liste de float, [latitude, longitude, timezone]
- `"sky"` : description du ciel :
    - `"turtle46"` : ciel soc à 46 directions en forme de "tortue"
    - `["file", filepath]` : ciel décrit par le fichier `filepath` (string)
    - `[nb_azimut, nb_zenith, "soc" ou "uoc"]`

- rayonnements à activer par un booléen:
    - `"direct"`
    - `"diffus"`
    - `"reflected"`
- `"infinite"` : booléen, active ou non la reproduction infinie de la scène
- paramètres de réflectance :
    - si CARIBU : `"caribu opt"` : dict tel que `"band" = tuple`
      1-tuple : définie la réflectance, objet opaque
      2-tuple : définie la réflectance et la transmittance d'un objet transparent symétrique
      4-tuple : définie la réflectance et la transmittance des faces d'un objet transparent asymétrique
    - si RATP : `"reflectance coefficients"` : liste, pour chaque espèce de plantes en entrée on définit une liste de réflectance pour chaque longeur d'onde

      *par exemple* : `[[0.1, 0.2]]` correspond à une réflectance du PAR de 0.1 et une réflectance de 0.2 du NIR pour l'espèce 0.

### Valeurs par défaut :

```
environment["coordinates"] = [46.4, 0. , 1.]
environment["sky"] = "turtle46"
environment["diffus"] = True
environment["direct"] = True
environment["reflected"] = False
environment["infinite"] = False
environment["reflectance coefficients"] = []
environment["caribu opt"] = {}
```

## Description de `lightmodel_parameters`

## CARIBU

### Clés d'entrée

- `"sun algo"` : `"rapt"` ou `"caribu"`, quelle méthode de calcul pour obtenir la position du soleil en fonction d'une date
- `"debug"` : booléen, active la description de CARIBU
- `"sensors"` : liste, décrit la position d'un ou plusieurs capteurs
    - `["grid", [dx, dy, dz], [nx, ny, nz], [Ox, Oy, Oz], folderoutpath, "vtk"]` : permet de construire une grille de capteurs (plaque horizontale), dxyz fournit les dimensions d'un capteur, nxyz le nombre de capteurs sur chaque axe, Oxyz l'origine de la grille. L'élément `"vtk"` est optionnelle et permet d'imprimer les résultats des capteurs au format VTK dans le dossier `folderoutpath`.

### Valeurs par défaut

```
lightmodel_parameters["sun algo"] = "caribu"
"debug" not in lightmodel_parameters
"sensors" not in lightmodel_parameters
```

## RATP

### Clés d'entrée

- `"voxel size"` :
    - `[float, float, float]` : donne les dimensions d'un voxel pour chaque axe
    - `"dynamic"` : définit des voxels carrés tels qu'un côté $\delta l = 5 \times Aire_{triangle_{max}}$
- `"soil reflectance"` : `[float, float]`, paramètre de réflectance du sol pour une ou plusieurs longueurs d'onde, chaque entrée de la liste correspond à une longueur d'onde.
- `"mu"` : liste de float, défini un paramètre de clumping pour chaque espèce en entrée
- `"tesselation level"` : int, si on a une triangulation en entrée, subdivise n fois les triangles à cheval entre plusieurs voxels

- `"angle distrib algo"` : méthode pour obtenir la distribution d'angles des feuilles :
    - `"compute global"` calcule la distribution par espèce
    - `"compute voxel"` calcule la distribution localement au voxel et par espèce
    - `"file"` : distribution lue dans un fichier
- `"nb angle classes"` : int, nombre de classes d'angles si la distribution est calculée dynamiquement
- `"angle distrib file"` : chemin du fichier contenant la distribution d'angle
- `"origin"` : possibilité de définir l'origine de la grille manuellement
    - `[xorigin, yorigin]` et $zorigin = -zmin$
    - `[xorigin, yorigin, zorigin]`
- `"number voxels"` : `[nx, ny, nz]`, possibilité de définir manuellement les dimensions de la grille, nombre de voxels sur chaque axe

**Valeurs par défaut**

```
lightmodel_parameters["voxel size"] = [0.1, 0.1, 0.1]
lightmodel_parameters["soil reflectance"] = [0., 0.]
lightmodel_parameters["mu"] = [1.]
lightmodel_parameters["tesselation level"] = 0
lightmodel_parameters["angle distrib algo"] = "compute global"
lightmodel_parameters["nb angle classes"] = 9
if "origin" not in  lightmodel_parameters : orig = [xmin, ymin, -
zmin]
if "number voxels"not in lightmodel_parameters : nxyz selon les
dimensions de la scène
```

## Description de `geometry`

### Clés d'entrée

- `"scenes"` : liste, chaque entrée correspond à une espèce. Type des entrées
    - Scene PlantGL
    - fichier VGX

        - dictionnaire décrivant une grille de voxels:
            - `"LA"` : correspond à m_lais dans l-egume,  un tableau au format `scipy.array`, de dimension `[nespèces, nz, ny, nx]` et indique pour chaque voxel et pour chaque espèce une surface foliaire
            - `"distrib"` : correspond à ls_dif dans l-egume, liste de `scipy.array`, correspond à la distribution d'angles foliaires pour chaque espèce
- `"transformations"` : dictionnaire contenant des modifications à effectuer sur une ou des scènes d'entrée (entrée optionnelle)
    - `"rescale"` : liste de float, une entrée par espèce, coefficient d'agrandissement de la scène
    - `"translate"` : liste de Vector3, un vecteur de translation par espèce
    - `"xyz orientation"` : liste de string, description de la convention d'orientation pour chaque espèce d'entrée et effectue une rotation de la scène pour ramener à la convention x+ = Nord. Description possible :
        - `"x+ = S"` : x+ correspond au Sud
        - `"x+ = W"` : x+ correspond à l'Ouest
        - `"x+ = E"` : x+ correspond à l'Est
        - `"y+ = y-"` : on inverse l'axe y
    - `"scenes unit"` : liste de string, description de l'échelle de mesure pour chaque espèce et effectue un agrandissement si elle est différente de l'unité de la scène finale. Entrée possible : `"mm"`, `"cm"`, `"m"`, `"dam"`, `"hm"`, `"km"`
- `"stems id"` : liste de tuple, `(indice de la shape/organe, indice de l'espèce)`
- `"domain"` : correspond au pattern dans le plan xy à reproduire dans CARIBU `((xmin, ymin), (xmax, ymax))`

### Valeurs par défaut

Hormis la géométrie en entrée, ces paramètres sont optionnels et sont absents par défaut.

## Notes sur les méthodes de l'outil

Constructeur

Construction de la scène géométrique finale

Calcul du rayonnement

Transfert des résultats vers les modèles FSP

#### CN-Wheat

#### l-egume

Autres méthodes utiles

#### Ecriture VTK

#### Analyse d'une triangulation par des outils tiers