
OpenAlea

Release 1.0.0a

Jul 23, 2019

1	Installation	3
2	Tutorials and Gallery	5
3	Packages	7
4	Development	9
5	User Guide	11
5.1	Visual Programming	11
5.2	Python Scripting	11
6	Installation	13
6.1	Conda Installation	13
6.2	OpenAlea Installation	13
7	Application Domains	15
8	Tutorials	17
8.1	Using Visualea : Beginning	17
8.2	Using Visualea : Weberpenn	27
9	Gallery	33
10	Packages	35
10.1	Modelling with OpenAlea	35
10.2	Plant Models	38
10.3	Plant Biophysics	38
10.4	Phenotyping	39
11	Development	41
11.1	Contributing	41
11.2	Moving from Python 2 to Python 3	43
12	License	45
13	Help	47



You can find here basic guides and procedures for using **OpenAlea** with instructions on how to develop your own package.

CHAPTER 1

Installation

For general information about installation, please read [Installation Guide](#).

If you are looking for a specific package installation, refer to the package documentation in [Packages](#).

CHAPTER 2

Tutorials and Gallery

You can find examples of what can be done with **OpenAlea**, you can check the [Tutorials](#) and [Gallery](#) pages.

CHAPTER 3

Packages

[Packages](#) page also contains brief presentations of official and available packages.

CHAPTER 4

Development

If you want to contribute to the project, you are welcome. See the [Development](#) page for more information.

Models under OpenAlea are in the form of components that could be used as such, or combined together to build user-customised applications. OpenAlea provides two different ways to interact with components:

- by visual programming, using Visualea
- by writing scripts, using a standard python development environment

Both methods allow you to save your application and run it routinely in batch mode. Visual programming is easier to start with, and it allows you to rapidly discover components of a package. Python scripting allows for programming more complex tasks, and provides an access to additional functionalities of models, by importing python modules that compose them.

5.1 Visual Programming

OpenAlea provides an high level visual programming interface Visualea.

The main documentation is on [Packages](#) at Visualea.

Tutorials have been made on Visualea. You can find them on [Tutorials](#).

5.2 Python Scripting

OpenAlea packages are available under the python scripting language. This allows to use the components of packages (as in visualea), but also to directly import python modules and get access to all functionalities documented in package's API.

5.2.1 Using Python

Python is a scripting language widely spread in the scientific community. It has a lot of advantages :

- It is easy to learn, even for non programmers.

- It is an high level language, based on the object paradigm
- It is extensible with external libraries
- Multi-platform (Linux, Windows, Mac)

You can learn the basics with the [Official Tutorial](#)

Python scripts could be launched by invoking python from a command line:

```
python myScript.py
```

You can also launch a Python interpreter and run or test your script step by step.

The default interpreter could be launch by typing python in a terminal (Linux, Mac) or in a DOS windows. Under Windows, you may also use one of the following method:

- Use Start Menu -> openalea -> python shell
- You can use the default python interpreter GUI : IDLE (start menu -> Python 2.4 -> IDLE)

You can also use more ergonomic user interface, like ipython, that provides usefull functionalities (completion, col-oration, execution of block of lines,...)

5.2.2 Importing OpenAlea Modules

The magic line which will make available all openalea modules is:

```
import openalea
```

All OpenAlea modules are available in the openalea namespace. Refer to the modules documentation to learn how to use them.

You can also write simple python scripts in order to execute the same code several times.

6.1 Conda Installation

Conda is a package manager that can be installed on Linux, Windows, and Mac. If you have not yet installed conda on your computer, follow these instructions : [Conda Installation](#).

6.2 OpenAlea Installation

The *recommended* way to install OpenAlea is to create a new conda environment.

First, create an environment named *openalea*:

Launch a console or a terminal (See Anaconda Prompt in Start menu on windows).

In this console, to install a given openalea package *<package_name>* with its dependencies, execute this::

```
conda create -n openalea -c openalea openalea.<package_name>
```

Here is an example if you want only *PlantGL*, *lpy*, *MTG* and *Caribu*:

```
conda create -n openalea -c openalea openalea.plantgl openalea.lpy openalea.mtg_
↪alinea.caribu boost=1.66
```

Activate the *openalea* environment:

```
conda activate openalea
```

In this environment, you may also want to install other Scientific Python packages:

```
conda install notebook=5.4 matplotlib pandas
```

In the documentation of each package, a installation procedure is described.

CHAPTER 7

Application Domains

Here are some interactive tutorials that can help you in your projects.

8.1 Using Visualea : Beginning

Here is a tutorial in which you will see how to implement a simple modeling problem in *Visualea*

Here is what you need for the following tutorial

```
conda create -n visualea_tuto -c openalea openalea.visualea openalea.components_
↪openalea.plantgl boost=1.66 -c openalea/label/unstable
conda activate visualea_tuto
```

Once you installed and activated the OpenAlea environment (see [Installation](#)), execute this

```
visualea
```

8.1.1 The Goal

We measured some tree data and saved these in a tabbed editor (like Excel). The data has been exported in a CSV file. We want to have a simple 3D representation of the measured tree.

Here is the data :

X	Y	crown_up	crown_bot	trunk_diameter
0	0	10	20	2
10	12	12	18	3
20	22	8	23	3.4
0	18	14	22	2.5

You may want to download the [CSV file](#).

8.1.2 Step 1 : Create Your Own Package

First of all, we need to create a package where to put your work (dataflow, node definition, data, ...). A package is in fact a simple directory containing python files.

Create a package

1. Select **Package Manager** -> **Add** -> **Package**
2. Fill the form :
 - **Name** : standbuilder
 - **Description** : build stand representation from measured data
 - **Version** : 0.1
 - **License** : Cecill-C
 - **Authors** : All collaborators and package writer
 - **Institutes** : ...
 - **URL** : ...
 - **Path** : /home/myhome/openalea_pkg (could be anywhere you want)
3. Click “OK”

Your new package should appear in the package manager.

Tip: The path corresponds to the directory where the python file will be written. Choose it carefully in order to be able to find it later.

8.1.3 Step 2 : Read CSV Data

Create a dataflow to read and view a file

Tip: Leaving the cursor on any item in the Package Manager, or on nodes or ports in the dataflow view brings up a tooltip. Clicking on them also shows some documentation in the “Help” tab (bottom-left-hand corner).

1. In the Package Manager tab (left column), open the *openalea.file* folder. You should see a list of nodes.

Note: You can search for a particular node in the Search tab.

2. In the Package Manager tab, drag the `read` node from the *openalea.file* package to the workshop. It should now appear on the canvas.

3. In the workspace, right click on the `read` node and choose “Open Widget”. Then browse for the “stand.csv” file (no need to validate anything, changes are automatically taken into account so you can simply close the window).
4. Drag the `text` node from the *openalea.data.structure.string* folder onto the workspace.
5. Connect the output of the `read` node to the input of the `text` node.

View the file contents

1. Right click on the `text` node and select “Run”
2. Right click on the `text` node and select “Open Widget”

Build a CSV object

In order to manipulate the CSV data, we are going to build a CSV object.

1. Select the search tab in the package manager
2. Type CSV
3. Drag the `read csv` node on the workspace
4. Do the same to create a `getitem` node (*openalea.python.method.getitem*)
5. Connect `read`’s output to `read csv`’s input
6. Connect `read csv`’s first output to `getitem`’s first input
7. Add an `int` node on the workspace, and connect its output to the second input of `getitem`
8. Execute the graph by selecting “Run” in the context menu of the `getitem` node
9. Print the output in the shell : Right click on the output port, and select “Print”

Save your work

1. Select **File** -> **Save as composite node** (CTRL + S)
2. In the selector dialog, click “New” Button
3. In the new dialog
 - Select the *standbuilder* package in the combo box
 - Enter the name : *readcsv_1*
 - Add a description : *Read data file*
 - Click “Ok”
4. In the selector, click “Ok” button
5. The new graph should appear in the *standbuilder* package.

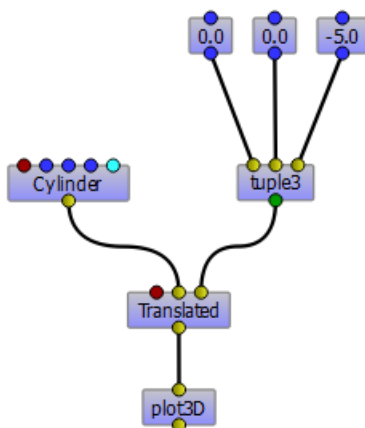
8.1.4 Step 3 : Create a simple 3D representation of one tree

Before displaying the whole stand, we must rebuild a tree. In this tutorial we build a very simple tree representation composed by a sphere for the crown and a cylinder for the trunk.

Create a 3D object

This simple dataflow shows how to display a scene object.

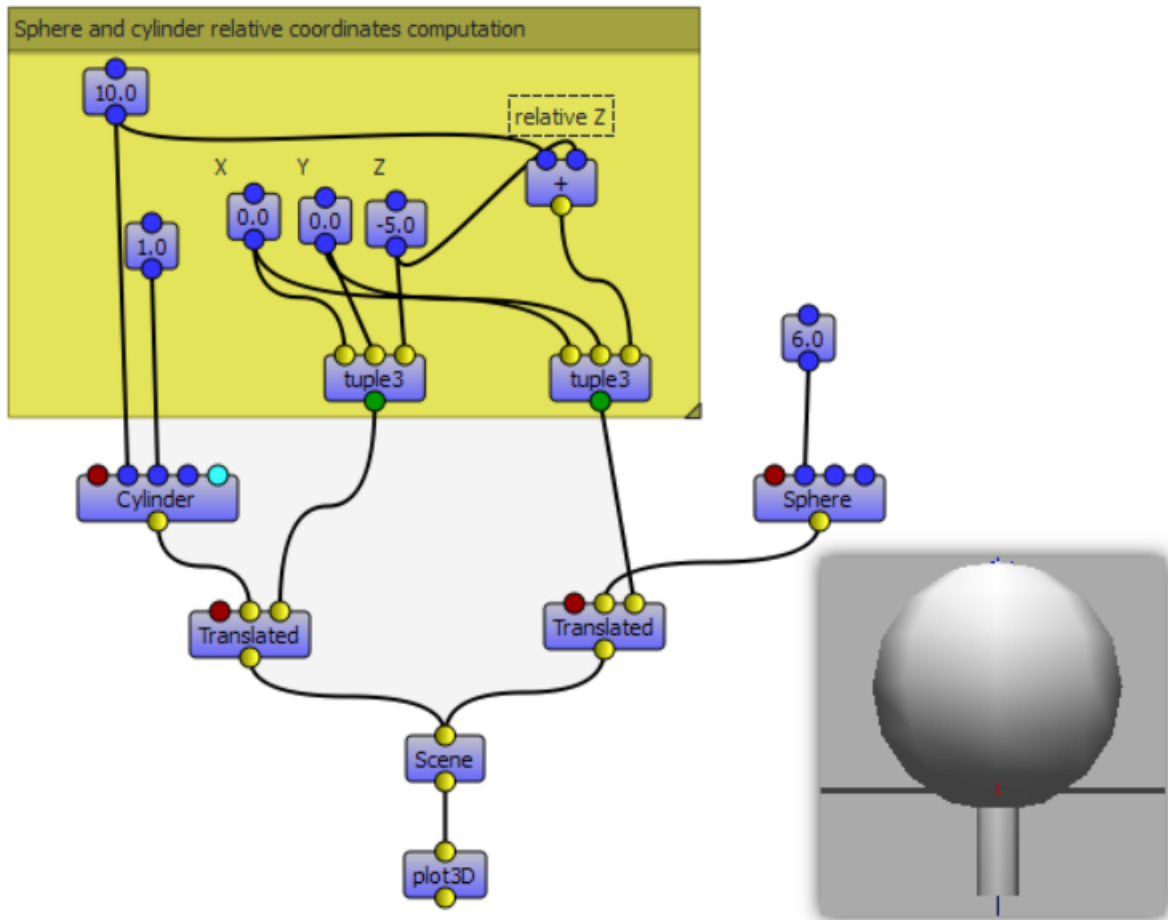
1. First step, we create a new workspace : Select **File -> New Empty Workspace** (CTRL+T)
2. Create the following dataflow by using PlantGL nodes
 - `vplants.plantgl.objects.cylinder` creates a cylinder
 - `vplants.plantgl.objects.translated` moves the input object
 - `openalea.data.structure.tuple.tuple3` to set the translation vector
 - `vplants.plantgl.visualization.plot3d` to view the result
 - `openalea.data.structure.float` to set the parameters of the tuple3 node



Create a simple tree

To build our tree, we must construct a PlantGL scene containing a cylinder and a sphere.

1. Modify the previous dataflow as follow:
 - Add a `vplants.plantgl.objects.sphere` object
 - Add a `vplants.plantgl.objects.translated` object
 - Add a `vplants.plantgl.objects.scene` object
 - Connect the 2 translated objects to a `vplants.plantgl.objects.scene` object
2. Save this dataflow in your standbuilder package as **simple_tree**



8.1.5 Step 4 : Create a Macro Node / Group Nodes

We will need to use the previous dataflow to build trees. To simplify this procedure, we would like to use a simple node and not a complex dataflow. For that we are going to embed the previous dataflow in a *composite node* (also named *macro node*).

Transform `simple_tree` to a reusable composite node

1. Select **simple_tree** in the package manager
2. Right click on the **simple_tree** graph, select “Properties” and click on the “Inputs / Outputs” button
3. Add 5 inputs with the + button :
 - X - IInt - 0 - X position
 - Y - IInt - 0 - Y position
 - crown_up - IFloat - 16.0 - Top of the crown
 - crown_bot - IFloat - 8.0 - Bottom of the crown
 - trunk_dia - IFloat - 3.0 - Trunk diameter
4. Add 1 output with the + button

- scene - None - PlanGL scene

I/O Configuration

Inputs

	Name	Interface	Value	Description
1	x	IInt	0	X position
2	y	IInt	0	Y position
3	crown_up	IFloat	16	Top of the crown
4	crown_bot	IFloat	8	Bottom of the crown
5	trunk_dia	IFloat	3	Trunk diameter

+

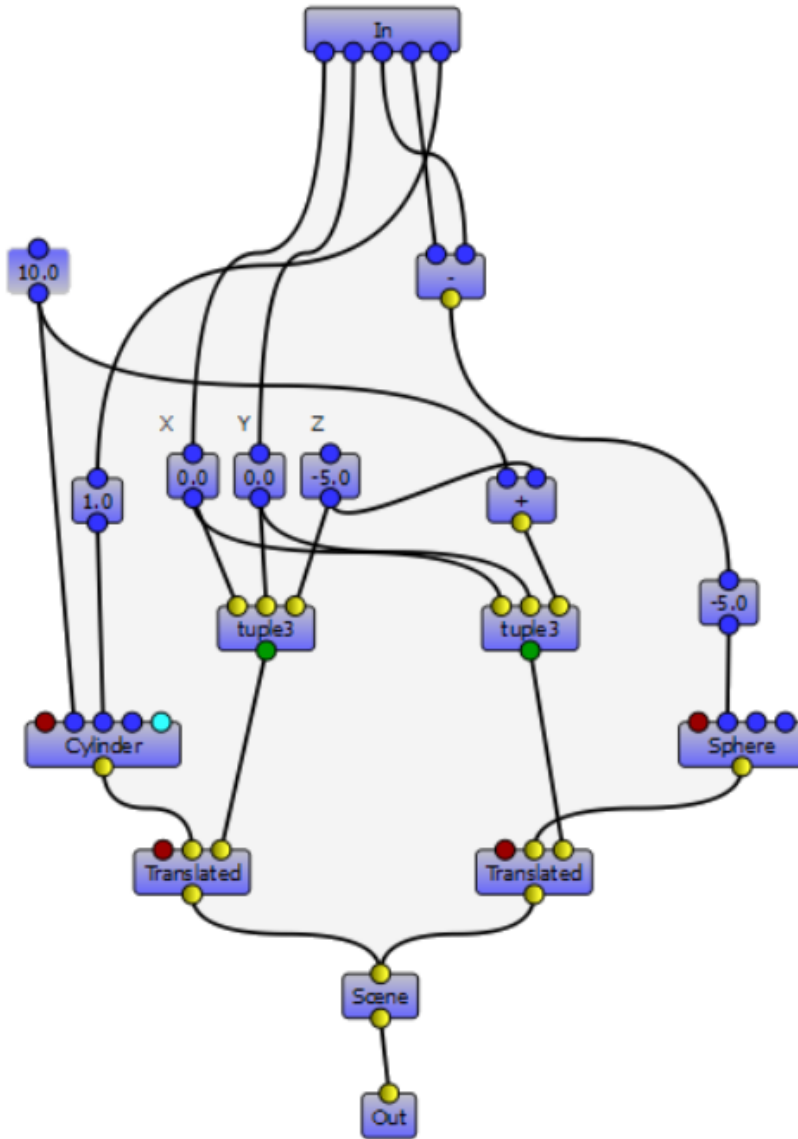
Outputs

	Name	Interface	Description
1	scene	None	PlanGL scene

+

OK Cancel

- Click “OK” and the buttons will appear in the workshop
- Modify the graph as follow
 - Connect input 0 and 1 to the X and Y nodes
 - Connect input 2 and 3 to a minus node `openalea.math.-`, and connect the result to the crown radius
 - Connect input 5 to the trunk radius
 - Connect input 3 to the crown bottom
- Save your work as a new composite node in standbuilder named **tree_scene**

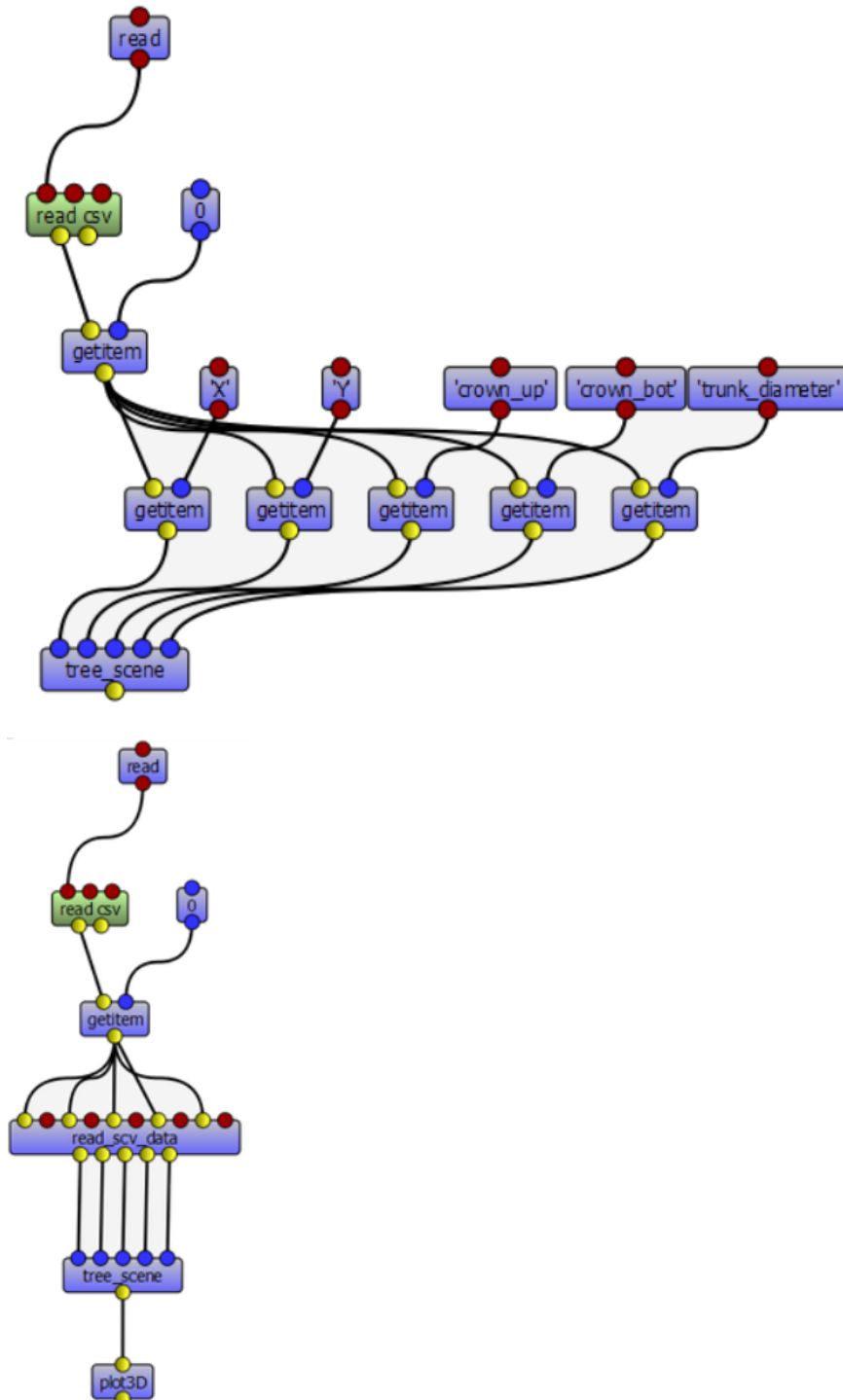


Using the new composite node in a dataflow

1. Open our first dataflow **readcsv_1** in the standbuilder package (doubleclick)
2. Drag the node `standbuilder.tree_scene` on the new workspace
3. Add 5x `getitem` and 5x `string` object
4. Connect the nodes as the picture in order to retrieve to different object properties
5. Add a `plangl.visualization.plot3D` object and connect it to the output of `tree_scene`
6. Run the dataflow several times and change the value of the first `getitem` (object index)
7. Save the dataflow in the standbuilder package as **readcsv_2**

Create a composite node by grouping nodes

1. Select the 5 `getitem` and their associated `string` object
2. Click on Menu **Workspace** -> **group** (CTRL+G)
3. Run the dataflow
4. Save it in the standbuilder package as `readcsv_3`

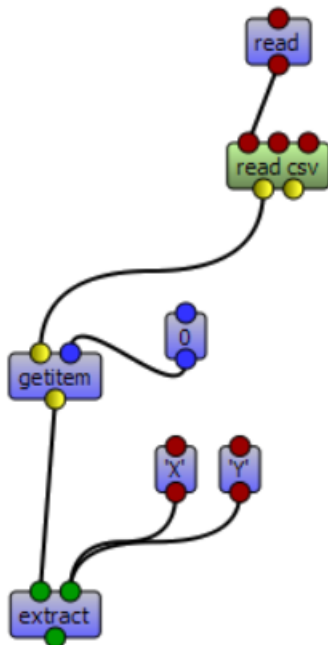


8.1.6 Step 5 : Get the spatial distribution of the trees

We want to extract from the csv object the X and Y properties and plot them in 2D.

Extract data

1. Create a new workspace (CTRL+T)
2. Add a `read` node and a `read csv` node to read a csv file
3. Set the file to read by opening the `read` widget (*Open Widget*)
4. Run and display the output (output port context menu -> *Print* or *Tooltip*) : it's a list of obj
5. Add a `getitem` node and an `int` node to select an object in the list
6. Add an `extract` node and 2 `string` nodes to select properties in a particular object
7. Set the 2 `string` objects to X and Y
8. Run and display the output (output port context menu -> *print* or *tooltip*) : it's a list containing the X and the Y properties of the selected object.

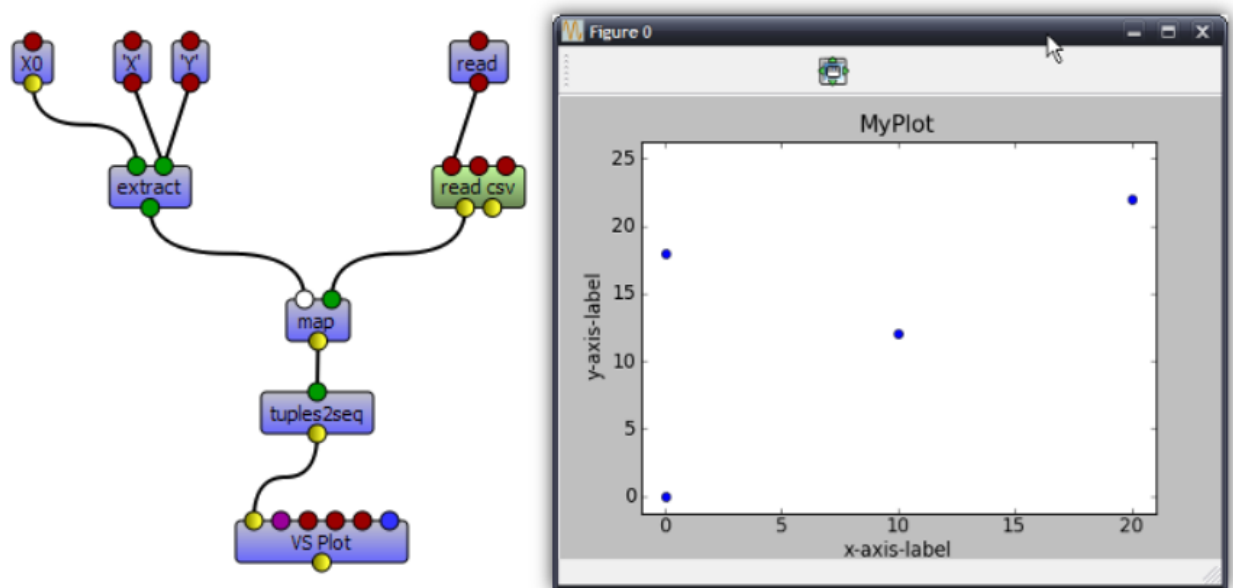


Implement iterative process

We want to do the same thing, but for all the CSV objects contained in the file.

1. Remove the `getitem` and the `int` nodes (with `suppr`)
2. Add an `openalea.function operator.map`
3. Connect the output of `extract` to the first input of `map`
4. Connect the output of `read csv` to the second input of `map`
5. Add an `openalea.flow control.X` node and connect its output the first input of `extract`

6. Run the map object and display the result



Note: The X object represents a function variable. The map apply a function to each element received in its second input.

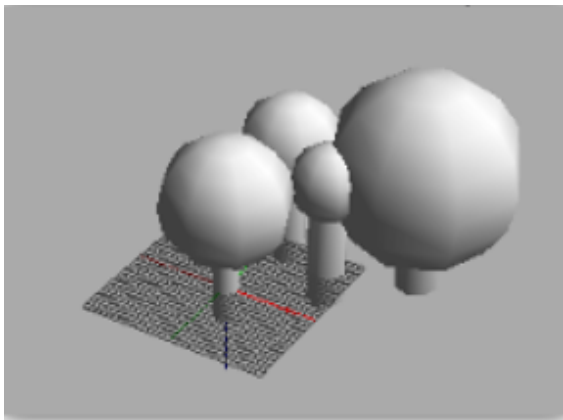
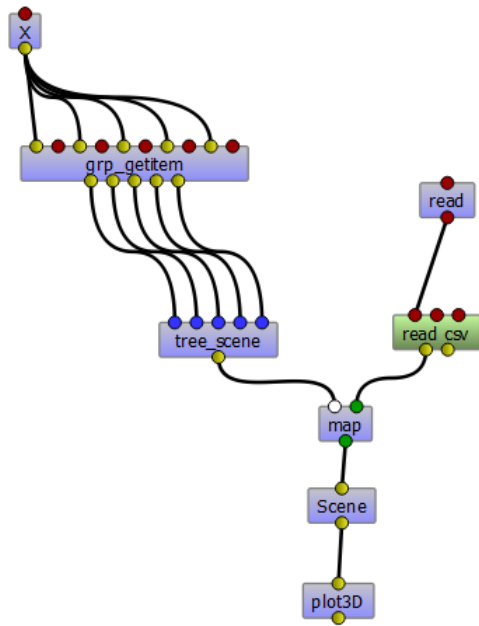
Plot 2D

1. Add the nodes `openalea.plottools.VS Plot` and `openalea.plottools.tuples2seq` on the workspace
2. Connect the map output the input of `tuples2seq` and the last output the `VS Plot` node.
3. Run the dataflow
4. Save it in the standbuilder package as **plot_csv**

8.1.7 Step 6 : Apply the process to multiple trees

In this step, we used the same method to build the entire stand

1. Open the `readcsv_3` dataflow
2. Modify it in order to plot in 3D all the tree contained in the file and not only one
3. Use a `openalea.flow control.X` node and a `openalea.functional.map` node
4. Save this work in your standbuilder package as **plot_stand**



8.2 Using Visualea : Weberpenn

8.2.1 Context

In *OpenAlea*, different tree architectures can be generated procedurally. *OpenAlea.WeberPenn* is based on the tree generating algorithm defined by Weber and Penn in 1995.

The model generates a tree structure based on a set of allometric rules. Fundamental parameters are, for instance, the overall appearance of the tree, the size of the lower part of the tree without axes, the max branching order or the curvature of the axes.

8.2.2 Install

Install Visualea and Weberpenn for this tutorial

```
conda install -c openalea openalea.weberpenn openalea.visualea
```

Then, execute this

```
visualea
```

8.2.3 Model Parameters

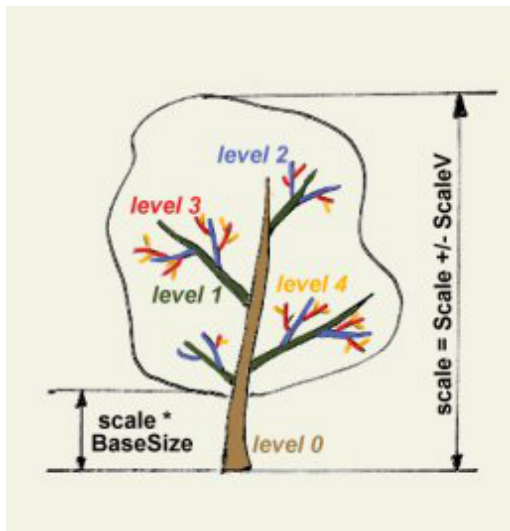
Image Courtesy of Wolfram Diestel, developer of the [Arbaro](#) software.

- General shape parameters :

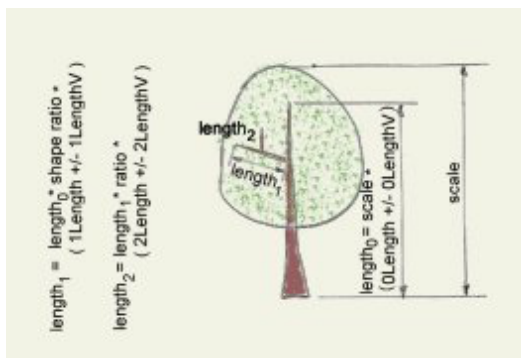
Scale and **ScaleV** : Global size of the tree

BaseSize - Size of the lower part of the tree without branches

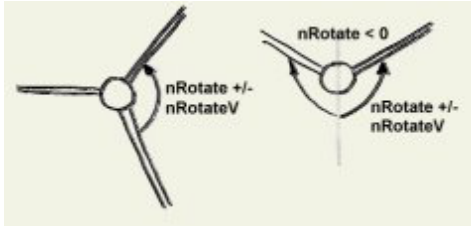
Each other parameters are defined for each branch level (or **order**) with order0 = trunk



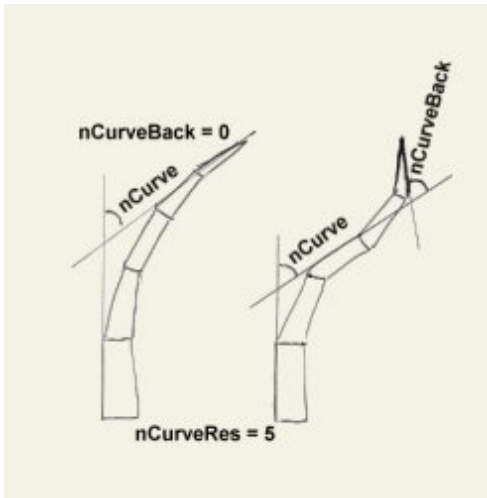
- Branch **length** is specified by the user at each order and is relative to the father branch length and to the overall shape of the tree.



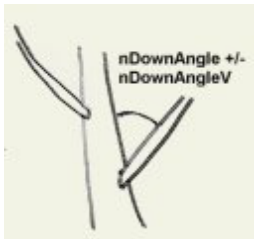
- rotation** (aka rotationV) define the phyllotaxis angle at each order



- **curve** parameter defines curvature of branches
curve back define inflexion angle

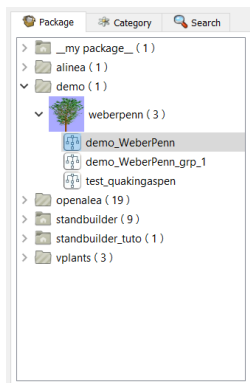


- **down_angle** define the angle of insertion between a branch and its father

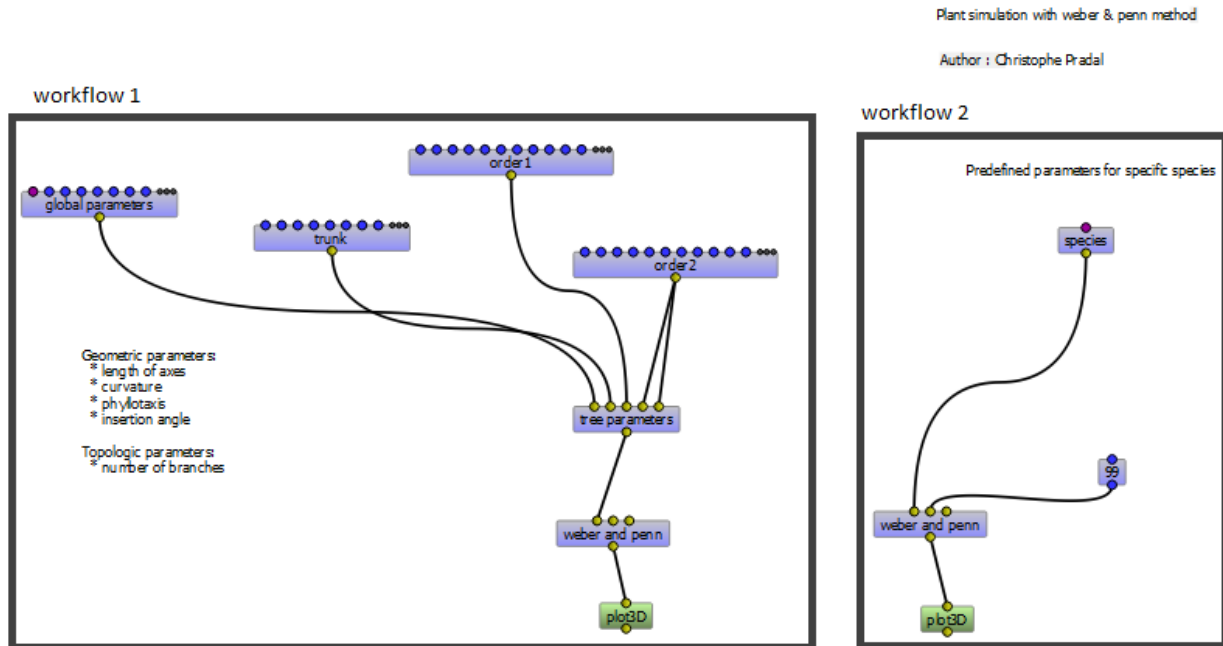


8.2.4 Begin with weberpenn

Once you've launched Visualea, in the package manager, go in *demo* and **double-click** on demo_WeberPenn.



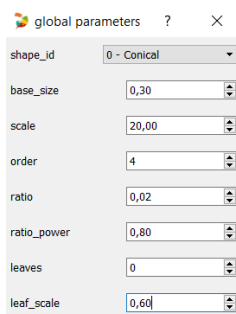
There will see two workflows in the workspace.



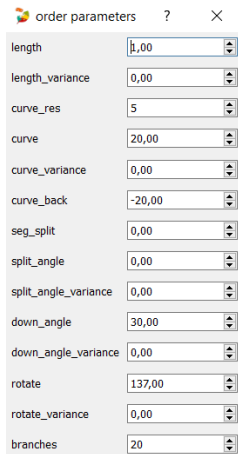
Workflow 1

On this workflow, there are the global parameters, trunk, order 1, order 2, tree parameters, weber and penn and plot3D nodes. You can change some parameters by **double-clicking** on the nodes.

- Global parameters change main parameters of the tree like its shape



- trunk, order 1 and order 2 allow to change parameters of the current order



Parameter	Value
length	1,00
length_variance	0,00
curve_res	5
curve	20,00
curve_variance	0,00
curve_back	-20,00
seg_split	0,00
split_angle	0,00
split_angle_variance	0,00
down_angle	30,00
down_angle_variance	0,00
rotate	137,00
rotate_variance	0,00
branches	20

- `tree parameters` synthesizes all the parameters into a unique *global parameters* object
- `weber` and `penn` computes the scene with all the generated surfaces
- `plot3D` displays a 3D-scene

Right-click on the `plot3D` node and **click** on “Run”. The scene will appear and you’ll be able to see the tree architecture corresponding to the inputs you’ve entered in the parameters nodes

Tip: You may want to change some parameters and see the impact on the tree architectures in real time. To do this, **right-click** on the `plot3D` node and **click** on “Mark as User Application” then run the node. Now, when you change a parameter, the scene updates instantly. Have fun !!!

Workflow 2

On this workflow, it is the same as the *Workflow 1* but you only have to choose the species you want in the `species` node. There are 3 species that have been preset.

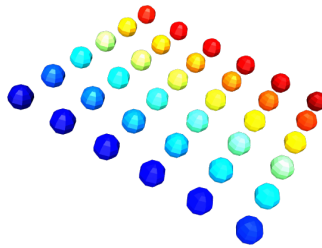


Fig. 1: 3D Geometric Modelling and Visualisation

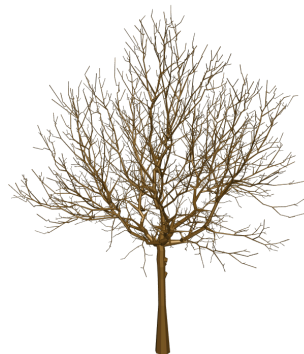


Fig. 2: 3D Reconstruction of Plant Architecture

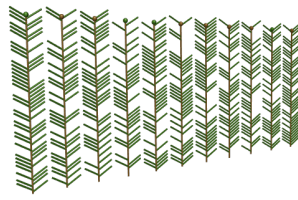


Fig. 3: L-Systems Simulation



Fig. 4: Reconstruction and Ecophysiology of Grapevine

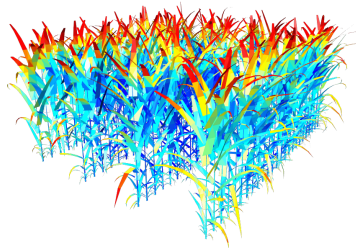


Fig. 5: Light Interception

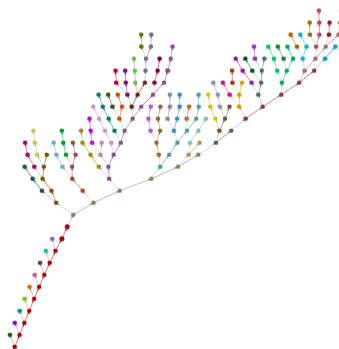
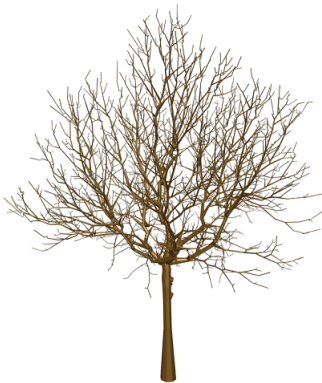


Fig. 6: MTG Visualisation and Data Extraction

10.1 Modelling with OpenAlea

10.1.1 MTG

Multiscale Tree Graph datastructure and interfaces



MTG package aims to define :

- A share data structure for plant architecture representation.
- Read and write MTG files.
- Export to various graph format.
- Several algorithms for MTG.

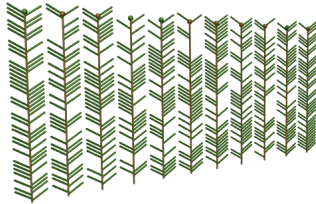
Authors : Christophe Pradal, Christophe Godin

Institutes : Cirad, Inria

Source Code : [Github](#)

10.1.2 L-Py

Plant simulation using Lindenmayer Systems with Python



L-systems were conceived as a mathematical framework for modeling growth of plants. L-Py is a simulation software that mixes L-systems construction with the Python high-level modeling language.

Authors : Frédéric Boudon, Christophe Pradal, Thomas Cokelaer, Przemyslaw Prusinkiewicz, Christophe Godin

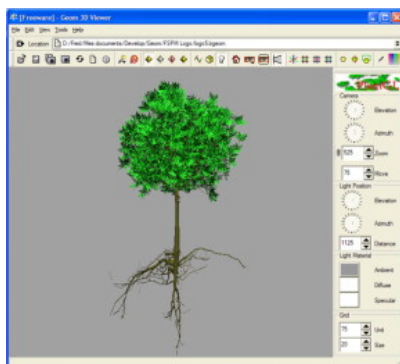
Institutes : Cirad, Inria, U. Calgary

Citation : Boudon et al., 2012, <https://doi.org/10.3389/fpls.2012.00076>

Source Code : [Github](#)

10.1.3 PlantGL

An open-source graphic toolkit for the creation, simulation and analysis of 3D virtual plants



Authors: Frédéric Boudon, Christophe Pradal, Christophe Nouguier, Jérôme Chopard, Christophe Godin

Institutes : Cirad, Inria

Citation : Pradal, Boudon et al., 2009, <https://doi.org/10.1016/j.gmod.2008.10.001>

Source Code : [Github](#)

10.1.4 OpenAlea Framework

OpenAlea Framework is able to discover and manage packages and logical components, build and evaluate dataflows and Generate final applications



Authors : Christophe Pradal, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier, Christophe Godin

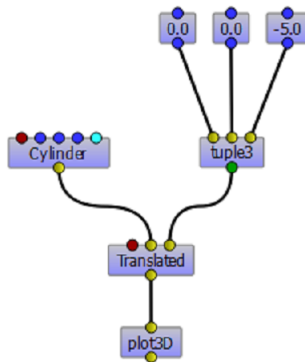
Institutes : Cirad, Inria, INRA

Citation : Pradal et al., 2008, <https://doi.org/10.1071/FP08084>

Source Code : [Github](#)

10.1.5 VisuAlea

An application that allows to use OpenAlea packages and to build dataflow graphically



Authors : Christophe Pradal, Samuel

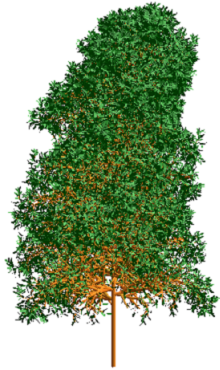
Institutes : Cirad, Inria

Source Code : [Github](#)

10.2 Plant Models

10.2.1 WeberPenn

An extension of the Weber & Penn model for OpenAlea



Authors : Christophe Pradal

Institutes : Cirad

Original Article : <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.380.2046>

Source Code : [Github](#)

10.3 Plant Biophysics

10.3.1 Hydroshoot

Hydroshoot model for 3D hydraulic architecture simulation



HydroShoot is a functional-structural plant modelling package taking into account hydraulic architecture and leaves energy budget and gas exchange.

Authors : Rami Albasha, Christian Fournier, Christophe Pradal

Institutes : INRA, Cirad

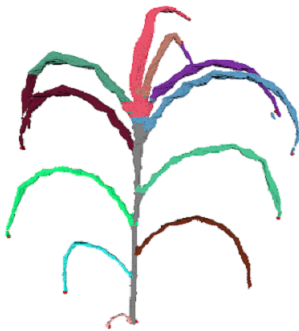
Citation : Albasha et al., 2019, <https://doi.org/10.1093/insilicoplants/diz007>

Source Code : [Github](#)

10.4 Phenotyping

10.4.1 Phenomenal

3D reconstruction from high-throughput plant phenotyping images



Plant high-throughput phenotyping aims at capturing the genetic variability of plant response to environmental factors for thousands of plants, hence identifying heritable traits for genomic selection and predicting the genetic values of allelic combinations in different environments.

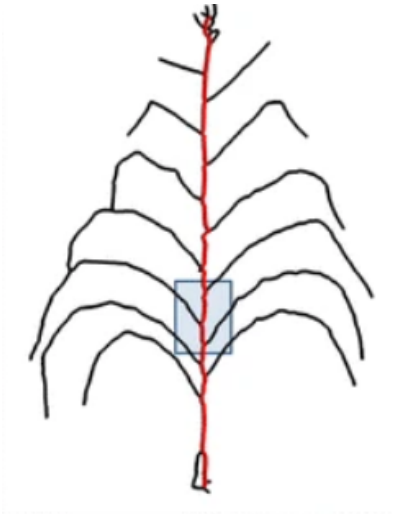
Authors : Simon Artzet, Christian Fournier, Christophe Pradal, Nicolas Brichet, Jerome Chopard, Michael Mielewczik

Institutes : INRA, Cirad

Source Code : [Github](#)

10.4.2 EarTrack

EarTrack is an imaging library to detect and track future position of ears on maize plants



Authors : Nicolas Brichet, Christian Fournier, Simon Artzet, Christophe Pradal,

Institutes : INRA, Cirad

Citation : Brichet et al., 2017, <https://doi.org/10.1186/s13007-017-0246-7>

Source Code : [Github](#)

Contents

- *Contributing*
 - *Introduction*
 - *How to contribute*
 - *How to make a proper bug report*
 - *Documentation*
- *Moving from Python 2 to Python 3*

11.1 Contributing

11.1.1 Introduction

OpenAlea is open-source. Everyone is welcome to contribute.

Note: All the packages are separated on two [Github](#) pages

- the official and stable packages on [OpenAlea](#)
- the unofficial, WIP and *to be reviewed* packages on [OpenAlea-Incubator](#)

There are many ways to contribute to [OpenAlea](#). The most common ways of contributing are coding or documenting different parts of the project. One may improve documentation which is as much important as improving the code itself. One could also create their own package, see [How to contribute](#).

Another way to contribute to the project is to report bugs and issues you are facing.

11.1.2 How to contribute

The main way to contribute is to fork the package repository you are interested in on [GitHub](#)

Note: Remember, the packages are found on different [GitHub](#) pages. The following steps describe a tutorial for an [OpenAlea](#) package. Make the good changes if you want to use [OpenAlea-Incubator](#) packages.

1. [Create an account](#) on GitHub if you do not already have one. You will choose your [GitHub](#) login `<your_login>`.
2. [Fork](#) the package repository of your choice (for instance, [MTG repository](#)) and click on the ‘Fork’ button near the top of the page. It generates a copy of the repository under your account on the [GitHub](#) user account. For more details on how to fork a repository see [this guide](#).
3. [Clone](#) your fork of the package repo from your [GitHub](#) account to your local disk

```
git clone https://github.com/<your_login>/<package_name>.git
cd <package_name>
```

4. Create a branch `<branch_name>` to hold your development changes

```
git checkout -b <branch_name>
```

and start making changes. Always use a feature branch. It’s good practice to never work on the master branch!

Note: In the above setup, your origin remote repository points to `<your_login>/<package_name>.git`. If you wish to fetch/merge from the main repository instead of your forked one, you will need to add another remote to use instead of origin. It’s good practice to choose the name `upstream` for it, and the command will be:

```
git remote add upstream https://github.com/openalea/<package_name>.git
```

If the package you chose come from [OpenAlea-Incubator](#), the command will be:

```
git remote add upstream https://github.com/openalea_incubator/<package_name>.git
```

And in order to fetch the new remote and base your work on the latest changes of it you can:

```
git fetch upstream
git checkout -b <your_name> upstream/master
```

5. Develop the feature on your feature branch on your computer, using Git to do the version control. When you’re done editing, add changed files using `git add` and then `git commit` files:

```
git add <modified_files>
git commit -m "description of what you've done"
```

to record your changes in Git, then push the changes to your [GitHub](#) account with:

```
git push -u origin <branch_name>
```

6. Once you’ve finished, you can create a **pull request** on the corresponding [GitHub](#). Follow [these](#) instructions to create a pull request from your fork.

11.1.3 How to make a proper bug report

11.1.4 Documentation

You can also contribute to the documentation. If you find some parts that are not explained enough or unclear, you can complete or improve the documentation.

Once you have forked the package on your device, you have to install [Sphinx](#) to generate the HTML output

```
pip install sphinx
```

In each package repository, it must be a `docs/` directory in which the reStructuredText documents are. You are pleased to modify or create these and generate the HTML output in the `docs/` directory

```
make html
```

Note: If you are creating your own package, you can build the [Sphinx](#) environment directly in the `docs/` directory

```
sphinx-quickstart
```

Once you are finished, you can add, commit and push what you have done on [GitHub](#) and then create a **pull request** (see [How to contribute](#)).

As we want all the documentation to look the same way, configure your Sphinx like us:

1. The theme we are using is the [Read the Docs Sphinx Theme](#). The theme can be installed like this

```
pip install sphinx_rtd_theme
```

Once you've installed the theme, write in your `conf.py` file

```
html_theme = "sphinx_rtd_theme"
```

Then write in the same file

```
html_theme_options = {
    'logo_only': True
}
```

2. Download the OpenAlea logo and put it your `_static` directory and then write in your `conf.py` file

```
html_static_path = ['_static']
html_logo = "_static/openalea_web.svg"
```

3. Mention the main website "openalea.rtd.io"

11.2 Moving from Python 2 to Python 3

CHAPTER 12

License

- The OpenAlea core is released under the Cecill-C license.
- The OpenAlea GUI, Visualea, is released under the Cecill-v2 license.

Note: Each external package can have its own license, please check it before using a package.

You can refer to the document [Package License Guidelines](#) for further explanations.

Please, [cite](#) the project if you use OpenAlea in your publications.

CHAPTER 13

Help

Need help ? Want to contact someone ? Contact prenon.nom@machin.fr.

CHAPTER 14

Indices and tables

- `genindex`
- `modindex`
- `search`