Phenomenal Documentation

Release 1.6.0

INRA / INRIA

Sep 27, 2018

Contents

A software framework for model-assisted analysis of high throughput plant phenotyping data

Contents

CHAPTER 1

What is Phenomenal ?

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 environment.

This first implies the automation of the measurement of a large number of traits to characterize plant growth, plant development and plant functioning. It also requires a fluent and versatile interaction between data and continuously evolving plant response models, that are essential in the analysis of the marker x environment interaction and in the integration of processes for predicting crop performance.

In the frame of the Phenome high throughput phenotyping infrastructure, we develop **Phenomenal**. A software framework dedicated to the analysis of high throughput phenotyping data and models.

Phenomenal currently consists of 2D image analysis workflows built with standard image libraries (VTK, OpenCV, Scikit.Image), algorithms for 3D reconstruction, segmentation and tracking of plant organs for maize (under development), and workflows for estimation of light interception by plants during their growth.

CHAPTER 2

Installation

2.1 Installation

Warning : Miniconda installation is strongly recommended

2.1.1 Installation with Miniconda (Windows, linux, OSX)

0. Install Miniconda

Follow official website instruction to install miniconda :

http://conda.pydata.org/miniconda.html

1. Install conda-build if not already installed

```
conda install conda-build
```

2. Create virtual environment and activate it

```
conda create --name phenomenal python
source activate phenomenal
```

3. Build and install openalea.phenomenal package

```
cd phenomenal/build_tools/conda
conda build -c conda-forge -c openalea .
conda install -c conda-forge -c openalea --use-local openalea.phenomenal
```

(Optional) Install several package managing tools :

```
conda install -c conda-forge notebook nose sphinx sphinx_rtd_theme pandoc coverage␣
↪ipyvolume nbconvert
```

2.1.2 Developer Install - Ubuntu (linux)

Warning : This installation procedure is not fully tested, We strongly recommand to install openalea.phenomenal with miniconda.

Contents

1. Install linux dependencies

Be sure opengl is installed on your machine

```
sudo apt-get update
sudo apt-get install freeglut3-dev
```

2. Miniconda installation

Follow official website instruction to install miniconda :

http://conda.pydata.org/miniconda.html

3. Create virtual environment and activate it

```
conda create --name phenomenal python
source activate phenomenal
```

4. Install dependencies with conda

```
conda install -c openalea/label/unstable -c openalea openalea.deploy openalea.core
conda install numba numpy scikit-learn scikit-image scipy matplotlib networkx vtk␣
↪opencv
```

(continues on next page)

(continued from previous page)

```
# Usefull tools for running example and documentation
conda install -c conda-forge nose notebook sphinx sphinx_rtd_theme pandoc ipyvolume

# On windows
conda install pywin32 [win]
```

2. Install openalea.phenomenal

```
git clone https://gitlab.inria.fr/phenome/phenomenal.git
cd phenomenal; python setup.py develop --prefix=$CONDA_PREFIX; cd ..
```

3. Test if installation is well installed (with nosetests package)

```
cd phenomenal
nosetests test
```

2.1.3 Windows

Warning :This installation procedure is not fully tested, We strongly recommand to install openalea.phenomenal with miniconda.

Contents

1. Install dependencies

```
# Basic
pip install numpy matplotlib scipy scikit-image

# Download Scipy, OpenCv, VTK, ... wheels on http://www.lfd.uci.edu/~gohlke/
↪pythonlibs/ and install it like this :
pip install *.whl

# Optional
pip install ipython ipython[notebook] nose

# OpenAlea.Deploy
git clone https://github.com/openalea/deploy
cd deploys; python setup.py install; cd ..

# OpenAlea.Core
```

(continues on next page)

(continued from previous page)

```
git clone https://github.com/openalea/core
cd core; python setup.py install; cd ..
```

2. Install openalea.phenomenal

```
git clone https://gitlab.inria.fr/phenome/phenomenal.git
cd phenomenal; python setup.py install; cd ..
```

3. Test if installation is well installed (with nosetests package)

```
cd phenomenal
nosetests test
```

2.1.4 Questions and Answers

Contents

How solve the server certificate verification failed problem on a ubuntu VM ?

```
export GIT_SSL_NO_VERIFY=1
#or
git config --global http.sslverify false
```

How solve the hostnanme on a ubuntu VM ?

cat /etc/host nano /etc/hostname

How to install Python-Irodsclient ?

```
git clone https://github.com/irods/python-irodsclient
cd python-irodsclient; python setup.py install --prefix=$CONDA_PREFIX; cd ..
```

How launch a Notebook Server on a cloud VM ?

```
jupyter notebook --no-browser --ip=<local_ip> &
disown
```

CHAPTER 3

Documentation

3.1 Tutorial Jupyter Notebooks

Tutorial Jupyter Notebooks are available on the git repository in the folder examples.

3.2 API References

3.2.1 API References

Release 1.6.0

Date Sep 27, 2018

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

openalea.phenomenal.data
openalea.phenomenal.image
openalea.phenomenal.calibration
openalea.phenomenal.object
openalea.phenomenal.display
openalea.phenomenal.multi_view_reconstruction
openalea.phenomenal.mesh
openalea.phenomenal.segmentation

Acces to plant's data

path_bin_images([plant_number])	According to the plant number return a dict[id_camera][angle] containing filename of the binary image.
path_raw_images([plant_number])	According to the plant number return a dict[id_camera][angle] containing filename of the raw image.
path_chessboard_images([plant_number])	According to the plant number return a dict[id_camera][angle] containing filename of the raw image.
raw_images([plant_number])	According to the plant number return a dict[id_camera][angle] of numpy array of the loader raw image.
bin_images([plant_number])	According to the plant number return a dict[id_camera][angle] of numpy array of the loader binary image.
chessboard_images([plant_number])	According to the plant number return a dict[id_camera][angle] of numpy array of the loader binary image.
chessboards([plant_number])	According to the plant number return a dict[id_camera] of camera calibration object :param plant_number: number of the plant desired (int) :return: dict[id_camera] of camera calibration object
calibrations([plant_number])	According to the plant number return a dict[id_camera] of camera calibration object
voxel_grid([plant_number, voxels_size])	According to the plant number and the voxel size desired return the voxel_grid of the plant.
tutorial_data_binarization_mask()	Return the list of required images to process the notebook tutorial on binarization.

openalea.phenomenal.data.path_bin_images

openalea.phenomenal.data.**path_bin_images**(*plant_number=1*)

According to the plant number return a dict[id_camera][angle] containing filename of the binary image.

Parameters **plant_number** (*int*) – Number of the plant desired

Returns **d** – dict[id_camera][angle] = filename

Return type dict of dict of string

openalea.phenomenal.data.path_raw_images

openalea.phenomenal.data.**path_raw_images**(*plant_number=1*)

According to the plant number return a dict[id_camera][angle] containing filename of the raw image. :param plant_number: number of the plant desired (int) :return: dict[id_camera][angle] of filename

openalea.phenomenal.data.path_chessboard_images

openalea.phenomenal.data.**path_chessboard_images**(*plant_number=1*)

According to the plant number return a dict[id_camera][angle] containing filename of the raw image. :param plant_number: number of the plant desired (int) :return: dict[id_camera][angle] of filename

openalea.phenomenal.data.raw_images

openalea.phenomenal.data.**raw_images**(*plant_number=1*)

According to the plant number return a dict[id_camera][angle] of numpy array of the loader raw image. :param plant_number: number of the plant desired (int) :return: dict[id_camera][angle] of loaded RGB image

openalea.phenomenal.data.bin_images

openalea.phenomenal.data.**bin_images**(*plant_number=1*)

According to the plant number return a dict[id_camera][angle] of numpy array of the loader binary image. A binary image is a numpy array of uint8 type. :param plant_number: number of the plant desired (int) :return: dict[id_camera][angle] of loaded grayscale image

openalea.phenomenal.data.chessboard_images

openalea.phenomenal.data.**chessboard_images**(*plant_number=1*)

According to the plant number return a dict[id_camera][angle] of numpy array of the loader binary image. A binary image is a numpy array of uint8 type. :param plant_number: number of the plant desired (int) :return: dict[id_camera][angle] of loaded grayscale image

openalea.phenomenal.data.chessboards

openalea.phenomenal.data.**chessboards**(*plant_number=1*)

According to the plant number return a dict[id_camera] of camera calibration object :param plant_number: number of the plant desired (int) :return: dict[id_camera] of camera calibration object

openalea.phenomenal.data.calibrations

openalea.phenomenal.data.**calibrations**(*plant_number=1*)

According to the plant number return a dict[id_camera] of camera calibration object

Parameters **plant_number** – number of the plant desired (int)

Returns dict[id_camera] of camera calibration object

openalea.phenomenal.data.voxel_grid

openalea.phenomenal.data.**voxel_grid**(*plant_number=1*, *voxels_size=4*)

According to the plant number and the voxel size desired return the voxel_grid of the plant.

Parameters

- **plant_number** – number of the plant desired (int)
- **voxels_size** – diameter of each voxel in mm (int)

Returns voxel_grid object

openalea.phenomenal.data.tutorial_data_binarization_mask

openalea.phenomenal.data.**tutorial_data_binarization_mask**()

Return the list of required images to process the notebook tutorial on binarization. The images are already load with opencv in unchanged format. images = [“mask_hsv.png”, “mask_clean_noise.png”, “mask_mean_shift.png”]

Returns list of image

Synthetic data (for test)

bin_images_with_circle([shape_image, ...])	
build_cube(cube_size, voxels_size, ...)	

openalea.phenomenal.data.bin_images_with_circle

openalea.phenomenal.data.**bin_images_with_circle**(*shape_image=(2454, 2056), circle_position=(1227, 1028), circle_radius=100*)

openalea.phenomenal.data.build_cube

openalea.phenomenal.data.**build_cube**(*cube_size*, *voxels_size*, *voxels_position*)

Image Methods

Threshold

threshold_meanshift(image, mean_image[, ...])	Threshold pixels in numpy array such as.
threshold_hsv(image, hsv_min, hsv_max[, mask])	Binarize HSV image with hsv_min and hsv_max parameters.
mean_image(images)	Compute the mean of a image list.
phenoarch_side_binarization(image, mean_image)	

openalea.phenomenal.image.threshold_meanshift

openalea.phenomenal.image.**threshold_meanshift**(*image*, *mean_image*, *threshold=0.3*, *reverse=False*, *mask=None*)

Threshold pixels in numpy array such as:

```
image / mean <= (1.0 - threshold)
```

If reverse is True (Inequality is reversed):

```
image / mean <= (1.0 + threshold
```

Parameters

- **image** (*numpy.ndarray of integers*) – 3-D array
- **mean_image** (*numpy.ndarray of the same shape as 'image'*) – 3-D array 'mean_image'
- **threshold** (*float, optional*) – Threshold value. Must between 0.0 and 1.0
- **reverse** (*bool, optional*) – If True reverse inequality
- **mask** (*numpy.ndarray, optional*) – Array of same shape as *image*. Only points at which mask == True will be thresholded.

Returns **out** – Thresholded binary image

Return type numpy.ndarray

See also:

`get_mean_image()`, *threshold_hsv()*

openalea.phenomenal.image.threshold_hsv

openalea.phenomenal.image.**threshold_hsv**(*image*, *hsv_min*, *hsv_max*, *mask=None*)

Binarize HSV image with hsv_min and hsv_max parameters. => cv2.inRange(hsv_image, hsv_min, hsv_max)

If mask is not None : => cv2.bitwise_and(binary_hsv_image, mask)

Parameters

- **image** (*numpy.ndarray of integers*) – 3-D array of image RGB
- **hsv_min** (*tuple of integers*) – HSV value of minimum range
- **hsv_max** (*tuple of integers*) – HSV value of maximum range
- **mask** (*numpy.ndarray, optional*) – Array of same shape as *image*. Only points at which mask == True will be thresholded.

Returns **out** – Thresholded binary image

Return type numpy.ndarray

See also:

threshold_meanshift()

openalea.phenomenal.image.mean_image

openalea.phenomenal.image.**mean_image**(*images*)

Compute the mean of a image list.

Parameters **images** (*[numpy.ndarray of integers]*) – list of 3-D array

Returns **out** – Mean of the list image

Return type numpy.ndarray

See also:

threshold_meanshift()

openalea.phenomenal.image.phenoarch_side_binarization

openalea.phenomenal.image.**phenoarch_side_binarization**(*image*, *mean_image*, *threshold=0.3*, *dark_background=False*, *hsv_min=(30, 25, 0)*, *hsv_max=(150, 254, 165)*, *mask_mean_shift=None*, *mask_hsv=None*)

Image Skeleton

skeletonize_thinning(image)	Thinning is used to reduce each connected component in a binary image to a single-pixel wide skeleton
skeletonize_erode_dilate(image)	Erode and dilate image to build skeleton

openalea.phenomenal.image.skeletonize_thinning

openalea.phenomenal.image.**skeletonize_thinning**(*image*)

Thinning is used to reduce each connected component in a binary image to a single-pixel wide skeleton

Parameters **image** – binary image with 0 or 255

Returns skeleton of a binary image.

openalea.phenomenal.image.skeletonize_erode_dilate

openalea.phenomenal.image.**skeletonize_erode_dilate**(*image*)

Erode and dilate image to build skeleton

Parameters **image** – binary image with 0 or 255

Returns skeleton of a binary image.

Morphologic Operation

dilate_erode(binary_image[, kernel_shape, ...])	Applied a morphology (dilate & erode) on binary_image on a ROI.
erode_dilate(binary_image[, kernel_shape, ...])	Applied a morphology (erode & dilate) on binary_image on mask ROI.
close(binary_image[, kernel_shape, mask])	Applied a morphology close on binary_image on mask ROI.

openalea.phenomenal.image.dilate_erode

openalea.phenomenal.image.**dilate_erode**(*binary_image*, *kernel_shape=(3, 3)*, *iterations=1*, *mask=None*)

Applied a morphology (dilate & erode) on binary_image on a ROI.

Parameters

- **binary_image** (*numpy.ndarray*) – 2-D array
- **kernel_shape** (*(N, M) of integers, optional*) – kernel shape of (dilate & erode) applied to binary_image
- **iterations** (*int, optional*) – number of successive iteration of (dilate & erode)
- **mask** (*numpy.ndarray, optional*) – Array of same shape as *image*. Only points at which mask == True will be processed.

Returns **out** – Binary Image

Return type numpy.ndarray

openalea.phenomenal.image.erode_dilate

openalea.phenomenal.image.**erode_dilate**(*binary_image*, *kernel_shape=(3, 3)*, *iterations=1*, *mask=None*)

Applied a morphology (erode & dilate) on binary_image on mask ROI.

Parameters

- **binary_image** (*numpy.ndarray*) – 2-D array
- **kernel_shape** (*(N, M) of integers, optional*) – kernel shape of (erode & dilate) applied to binary_image
- **iterations** (*int, optional*) – number of successive iteration of (erode & dilate)
- **mask** (*numpy.ndarray, optional*) – Array of same shape as *image*. Only points at which mask == True will be processed.

Returns **out** – Binary Image

Return type numpy.ndarray

openalea.phenomenal.image.close

openalea.phenomenal.image.**close**(*binary_image*, *kernel_shape=(7, 7)*, *mask=None*)

Applied a morphology close on binary_image on mask ROI.

Parameters

- **binary_image** (*numpy.ndarray*) – 2-D array
- **kernel_shape** (*(N, M) of integers*) – kernel shape of morphology close applied to binary_image
- **mask** (*numpy.ndarray, optional*) – Array of same shape as *image*. Only points at which mask == True will be processed.

Returns **out** – Binary Image

Return type numpy.ndarray

Input / Output

read_image(filename[, flags])	Read a image from a file name with opencv API.
write_image(filename, image)	Write a image in a file.

openalea.phenomenal.image.read_image

openalea.phenomenal.image.**read_image**(*filename*, *flags=-1*)

Read a image from a file name with opencv API.

Parameters

- **filename** – file name of the image
- **flags** –

Returns RGB or grayscale image

openalea.phenomenal.image.write_image

openalea.phenomenal.image.**write_image**(*filename*, *image*)

Write a image in a file.

Parameters

- **filename** – output filename where write the image
- **image** – numpy image to write

Returns None

Calibration

Target & Chessboard

Target()	
Chessboard([square_size, shape])	

openalea.phenomenal.calibration.Target

class openalea.phenomenal.calibration.**Target**

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__()	x.__init__(...) initializes x; see help(type(x)) for signature
add_image_points(camera_view, angle, image)	
get_3d_local_points()	
get_image_points()	

openalea.phenomenal.calibration.Chessboard

class openalea.phenomenal.calibration.**Chessboard**(*square_size=50*, *shape=(7, 7)*)

__init__(*square_size=50*, *shape=(7, 7)*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__([square_size, shape])	x.__init__(...) initializes x; see help(type(x)) for signature
detect_corners(id_camera, angle, image)	Detect chessboard corner in a image and save it in object with the id_camera and angle like keys.
dump(filename)	
get_corners_2d(id_camera)	
get_corners_local_3d()	
load(filename)	

Calibration

CalibrationCamera()	
CalibrationCameraTop()	
CalibrationCameraSideWith2TargetYXZ()	

openalea.phenomenal.calibration.CalibrationCamera

class openalea.phenomenal.calibration.**CalibrationCamera**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__()	x.__init__(...) initializes x; see help(type(x)) for signature
arr_pixel_coordinates(points_3d, ...)	Compute image coordinates of a 3d point
camera_frame(pos_x, pos_y, pos_z, rot_x, ...)	
dump(filename)	
get_camera_frame()	
get_projection(alpha)	
get_projection2(alpha)	
load(filename)	
pixel_coordinates(point_3d, width_image, ...)	Compute image coordinates of a 3d point
pixel_coordinates_2(point_3d, cx, cy, fx, fy)	Compute image coordinates of a 3d point
target_frame(pos_x, pos_y, pos_z, rot_x, ...)	

openalea.phenomenal.calibration.CalibrationCameraTop

class openalea.phenomenal.calibration.**CalibrationCameraTop**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__()	x.__init__(...) initializes x; see help(type(x)) for signature
arr_pixel_coordinates(points_3d, ...)	Compute image coordinates of a 3d point
calibrate(ref_target_points_2d, ... [, ...])	
camera_frame(pos_x, pos_y, pos_z, rot_x, ...)	
dump(filename)	
find_parameters(number_of_repetition)	
fit_function(x0)	
get_camera_frame()	
get_projection(alpha)	
get_projection2(alpha)	
load(filename)	
pixel_coordinates(point_3d, width_image, ...)	Compute image coordinates of a 3d point
pixel_coordinates_2(point_3d, cx, cy, fx, fy)	Compute image coordinates of a 3d point
project_points_3d(points_3d)	
target_frame(pos_x, pos_y, pos_z, rot_x, ...)	

openalea.phenomenal.calibration.CalibrationCameraSideWith2TargetYXZ

class openalea.phenomenal.calibration.**CalibrationCameraSideWith2TargetYXZ**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__()	x.__init__(...) initializes x; see help(type(x)) for signature
arr_pixel_coordinates(points_3d, ...)	Compute image coordinates of a 3d point
calibrate(ref_target_1_points_2d, ... [, ...])	Find physical parameters associated with a camera (i.e.
camera_frame(pos_x, pos_y, pos_z, rot_x, ...)	
dump(filename)	
find_parameters(number_of_repetition)	
fit_function(x0)	
get_camera_frame()	
get_projection(alpha)	

Continued on next page

Table 14 – continued from previous page

get_projection2(alpha)	
get_target_1_projected(alpha, ...)	
get_target_1_ref_points_global_3d(alpha, ...)	
get_target_2_projected(alpha, ...)	
get_target_2_ref_points_global_3d(alpha, ...)	
load(filename)	
pixel_coordinates(point_3d, width_image, ...)	Compute image coordinates of a 3d point
pixel_coordinates_2(point_3d, cx, cy, fx, fy)	Compute image coordinates of a 3d point
target_frame(pos_x, pos_y, pos_z, rot_x, ...)	

Frame

Frame([axes, origin])	A helping class to deal with change of referential in 3D space.
x_axis	
y_axis	
z_axis	

openalea.phenomenal.calibration.Frame

class openalea.phenomenal.calibration.**Frame**(*axes=None*, *origin=None*)

A helping class to deal with change of referential in 3D space.

__init__(*axes=None*, *origin=None*)

Constructor

Construct a new orthonormal Frame default is the canonical one.

Parameters

- ***axes*** **([array,array,array]) - orientation of axes each array is** the coordinate of the local axis in the global frame
- ***origin*** **(array) - position of the origin of this frame** in the global frame

Methods

__init__([axes, origin])	Constructor
arr_local_point(point)	Local coordinates of a global point
axis(i)	Return the coordinates of the ith axis of the frame
global_point(point)	Global coordinates of a local point
global_tensor(tensor)	Global coordinates of a local tensor
global_tensor2(tensor)	Global coordinates of a local tensor
global_vec(vec)	Global coordinates of a local vector
local_point(point)	Local coordinates of a global point
local_points(points)	Local coordinates of global points
local_tensor(tensor)	Local coordinates of a global tensor

Continued on next page

Table 16 – continued from previous page

local_tensor2(tensor)	Local coordinates of a global tensor expressed in the local plane Ox,Oy
local_vec(vec)	Local coordinates of a global vector
local_vecs(vecs)	Local coordinates of global vectors
origin()	Origin of this frame
rotation_to_global()	Return the inverse rotation associated with this frame
rotation_to_local()	Return the rotation associated with this frame

openalea.phenomenal.calibration.x_axis

openalea.phenomenal.calibration.**x_axis = array([1., 0., 0.])**

openalea.phenomenal.calibration.y_axis

openalea.phenomenal.calibration.**y_axis = array([0., 1., 0.])**

openalea.phenomenal.calibration.z_axis

openalea.phenomenal.calibration.**z_axis = array([0., 0., 1.])**

Data Structure

Image3D	
ImageView(image, projection[, inclusive, ...])	
VoxelGrid(voxels_position, voxels_size)	
VoxelSegment(polyline, voxels_position, ...)	
VoxelOrgan(label[, sub_label])	
VoxelSkeleton(segments, voxels_size)	
VoxelSegmentation(voxels_size)	

openalea.phenomenal.object.Image3D

class openalea.phenomenal.object.**Image3D**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

all([axis, out, keepdims])	Returns True if all elements evaluate to True.
any([axis, out, keepdims])	Returns True if any of the elements of *a* evaluate to True.
argmax([axis, out])	Return indices of the maximum values along the given axis.

Continued on next page

Table 18 – continued from previous page

argmin([axis, out])	Return indices of the minimum values along the given axis of *a*.
argpartition(kth[, axis, kind, order])	Returns the indices that would partition this array.
argsort([axis, kind, order])	Returns the indices that would sort this array.
astype(dtype[, order, casting, subok, copy])	Copy of the array, cast to a specified type.
byteswap([inplace])	Swap the bytes of the array elements
choose(choices[, out, mode])	Use an index array to construct a new array from a set of choices.
clip([min, max, out])	Return an array whose values are limited to `[min, max]`.
compress(condition[, axis, out])	Return selected slices of this array along given axis.
conj()	Complex-conjugate all elements.
conjugate()	Return the complex conjugate, element-wise.
copy([order])	Return a copy of the array.
cumprod([axis, dtype, out])	Return the cumulative product of the elements along the given axis.
cumsum([axis, dtype, out])	Return the cumulative sum of the elements along the given axis.
diagonal([offset, axis1, axis2])	Return specified diagonals.
dot(b[, out])	Dot product of two arrays.
dump(file)	Dump a pickle of the array to the specified file.
dumps()	Returns the pickle of the array as a string.
fill(value)	Fill the array with a scalar value.
flatten([order])	Return a copy of the array collapsed into one dimension.
getfield(dtype[, offset])	Returns a field of the given array as a certain type.
item(*args)	Copy an element of an array to a standard Python scalar and return it.
itemset(*args)	Insert scalar into an array (scalar is cast to array's dtype, if possible)
max([axis, out, keepdims])	Return the maximum along a given axis.
mean([axis, dtype, out, keepdims])	Returns the average of the array elements along given axis.
min([axis, out, keepdims])	Return the minimum along a given axis.
newbyteorder([new_order])	Return the array with the same data viewed with a different byte order.
nonzero()	Return the indices of the elements that are non-zero.
ones(shape[, voxels_size, world_coordinate, ...])	
ones_like(image_3d)	
partition(kth[, axis, kind, order])	Rearranges the elements in the array in such a way that the value of the element in kth position is in the position it would be in a sorted array.
prod([axis, dtype, out, keepdims])	Return the product of the array elements over the given axis
ptp([axis, out, keepdims])	Peak to peak (maximum - minimum) value along a given axis.
put(indices, values[, mode])	Set `a.flat[n] = values[n]` for all *n* in indices.
ravel([order])	Return a flattened array.
read_from_npz(filename)	
repeat(repeats[, axis])	Repeat elements of an array.

Continued on next page

Table 18 – continued from previous page

reshape(shape[, order])	Returns an array containing the same data with a new shape.
resize(new_shape[, refcheck])	Change shape and size of array in-place.
round([decimals, out])	Return *a* with each element rounded to the given number of decimals.
searchsorted(v[, side, sorter])	Find indices where elements of v should be inserted in a to maintain order.
setfield(val, dtype[, offset])	Put a value into a specified place in a field defined by a data-type.
setflags([write, align, uic])	Set array flags WRITEABLE, ALIGNED, (WRITEBACKIFCOPY and UPDATEIFCOPY), respectively.
sort([axis, kind, order])	Sort an array, in-place.
squeeze([axis])	Remove single-dimensional entries from the shape of *a*.
std([axis, dtype, out, ddof, keepdims])	Returns the standard deviation of the array elements along given axis.
sum([axis, dtype, out, keepdims])	Return the sum of the array elements over the given axis.
swapaxes(axis1, axis2)	Return a view of the array with *axis1* and *axis2* interchanged.
take(indices[, axis, out, mode])	Return an array formed from the elements of *a* at the given indices.
tobytes([order])	Construct Python bytes containing the raw data bytes in the array.
tofile(fid[, sep, format])	Write array to a file as text or binary (default).
tolist()	Return the array as a (possibly nested) list.
tostring([order])	Construct Python bytes containing the raw data bytes in the array.
trace([offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
transpose(*axes)	Returns a view of the array with axes transposed.
var([axis, dtype, out, ddof, keepdims])	Returns the variance of the array elements, along given axis.
view([dtype, type])	New view of array with the same data.
write_to_npz(filename)	
write_to_stack_image(folder_name)	
zeros(shape[, voxels_size, ...])	
zeros_like(image_3d)	

Attributes

T	Same as self.transpose(), except that self is returned if self.ndim < 2.
base	Base object if memory is from some other object.
ctypes	An object to simplify the interaction of the array with the ctypes module.
data	Python buffer object pointing to the start of the array's data.
dtype	Data-type of the array's elements.
flags	Information about the memory layout of the array.

Continued on next page

Table 19 – continued from previous page

flat	A 1-D iterator over the array.
imag	The imaginary part of the array.
itemsize	Length of one array element in bytes.
nbytes	Total bytes consumed by the elements of the array.
ndim	Number of array dimensions.
real	The real part of the array.
shape	Tuple of array dimensions.
size	Number of elements in the array.
strides	Tuple of bytes to step in each dimension when traversing an array.

openalea.phenomenal.object.ImageView

class openalea.phenomenal.object.**ImageView**(*image*, *projection*, *inclusive=False*, *image_ref=None*)

__init__(*image*, *projection*, *inclusive=False*, *image_ref=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(image, projection[, inclusive, ...])	x.__init__(...) initializes x; see help(type(x)) for signature

openalea.phenomenal.object.VoxelGrid

class openalea.phenomenal.object.**VoxelGrid**(*voxels_position*, *voxels_size*)

__init__(*voxels_position*, *voxels_size*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(voxels_position, voxels_size)	x.__init__(...) initializes x; see help(type(x)) for signature
bounding_box()	
from_image_3d(image_3d[, voxels_value, ...])	
read(filename)	
read_from_csv(filename)	
read_from_json(filename)	
read_from_npz(filename)	
read_from_xyz(filename, voxels_size)	
show()	
to_image_3d()	
volume()	Compute the volume of the voxel point cloud
write(filename)	
write_to_csv(filename)	

Continued on next page

Table 21 – continued from previous page

write_to_json(filename)	
write_to_npz(filename)	
write_to_xyz(filename)	

Attributes

voxels_position
voxels_size

openalea.phenomenal.object.VoxelSegment

class openalea.phenomenal.object.**VoxelSegment**(*polyline*, *voxels_position*, *closest_nodes*)

__init__(*polyline*, *voxels_position*, *closest_nodes*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(polyline, voxels_position, ...)	x.__init__(...) initializes x; see help(type(x)) for signature

openalea.phenomenal.object.VoxelOrgan

class openalea.phenomenal.object.**VoxelOrgan**(*label*, *sub_label=None*)

__init__(*label*, *sub_label=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(label[, sub_label])	x.__init__(...) initializes x; see help(type(x)) for signature
add_voxel_segment(voxels_position, polyline)	
get_highest_polyline()	
get_longest_segment()	
get_real_index_position_base()	
real_longest_polyline()	
voxels_position()	

openalea.phenomenal.object.VoxelSkeleton

class openalea.phenomenal.object.**VoxelSkeleton**(*segments*, *voxels_size*)

__init__(*segments*, *voxels_size*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(segments, voxels_size)	x.__init__(...) initializes x; see help(type(x)) for signature
to_voxel_grid()	
volume()	
voxels_position()	
voxels_position_polyline()	

openalea.phenomenal.object.VoxelSegmentation

class openalea.phenomenal.object.**VoxelSegmentation**(*voxels_size*)

__init__(*voxels_size*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__(voxels_size)	x.__init__(...) initializes x; see help(type(x)) for signature
get_growing_leafs()	
get_leaf_order(number)	
get_leafs()	
get_mature_leafs()	
get_number_of_leaf()	
VoxelSegmentation.get_plant_info	
get_stem()	
get_unknown()	
get_voxels_position([except_organs])	
read_from_json_gz(filename[, without_info])	
swap_leaf_order(number_1, number_2)	
write_to_json_gz(filename)	

Display

Image

show_image(image[, name_windows])	
show_images(images[, name_windows])	
show_image_with_chessboard_corners(image, ...)	
show_chessboard_3d_projection_on_image(...)	

openalea.phenomenal.display.show_image

openalea.phenomenal.display.**show_image**(*image*, *name_windows=''*)

openalea.phenomenal.display.show_images

openalea.phenomenal.display.**show_images**(*images*, *name_windows=''*)

openalea.phenomenal.display.show_image_with_chessboard_corners

openalea.phenomenal.display.**show_image_with_chessboard_corners**(*image*, *corners*, *name_windows=''*)

openalea.phenomenal.display.show_chessboard_3d_projection_on_image

openalea.phenomenal.display.**show_chessboard_3d_projection_on_image**(*image*, *points_2d_1*, *points_2d_2*, *figure_name=''*)

3D Data

Display([background_color])	
Scene()	

openalea.phenomenal.display.Display

class openalea.phenomenal.display.**Display**(*background_color=(1, 1, 1)*)

__init__(*background_color=(1, 1, 1)*)
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__([background_color])	x.__init__(...) initializes x; see help(type(x)) for signature
clean_all_actors()	
export_scene_to_vrml(file_prefix)	
export_scene_to_x3d(file_prefix)	
init_record_video(filename[, windows_size, ...])	
record_video(video_filename, elements, func)	
render([windows_size])	
reset_camera()	
screenshot(filename[, magnification])	
set_background_color(color)	
set_camera([elevation, azimuth, position, ...])	
show([windows_size, screenshot_filename, ...])	
switch_elements(elements, func)	

openalea.phenomenal.display.Scene

class openalea.phenomenal.display.**Scene**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

__init__()	x.__init__(...) initializes x; see help(type(x)) for signature
add_actor(actor)	
add_actor_from_arrow_vector(start_point, ...)	
add_actor_from_ball_position(position[, ...])	
add_actor_from_plane(center, normal[, ...])	
add_actor_from_text(text[, position, scale, ...])	
add_actor_from_vertices_faces(vertices, faces)	
add_actor_from_voxels(voxels_position, ...)	
add_actors(actors)	
add_actors_from_voxels_list(...[, colors])	
add_text_actor(text_actor)	
add_text_actors(text_actors)	
clean_all_actors()	
export_scene_to_vrml(file_prefix)	
export_scene_to_x3d(file_prefix)	
get_actor_from_arrow_vector(start_point, ...)	
get_actor_from_ball_position(position[, ...])	
get_actor_from_plane(center, normal[, ...])	
get_actor_from_text(text[, position, scale, ...])	
get_actor_from_vertices_faces(vertices, faces)	
get_actor_from_voxels(voxels_position, ...)	
init_record_video(filename[, windows_size, ...])	
record_video(video_filename, elements, func)	
render([windows_size])	
reset_camera()	
screenshot(filename[, magnification])	
set_background_color(color)	
set_camera([elevation, azimuth, position, ...])	

Continued on next page

Table 30 – continued from previous page

show([windows_size, screenshot_filename, ...])	
switch_elements(elements, func)	

Multi-View Reconstruction

Main's function

reconstruction_3d(image_views[, ...])	Construct a list of voxel represented object with positive value on binary image in images of images_projections.
project_voxel_centers_on_image(..., [, value, ...])	Create a image with same shape that shape_image and project each voxel on image and write positive value (255) on it.
project_voxels_position_on_image(...)	Create a image with same shape that shape_image and project each voxel on image and write positive value (255) on it.
image_error(img_ref, img_src[, precision])	Return false position and true negative result from the comparaison on two binaries images
reconstruction_error(voxels_grid, image_views)	Compute the reconstruction error (false positive and true negative) of the 3d reconstruction from the image view.

openalea.phenomenal.multi_view_reconstruction.reconstruction_3d

openalea.phenomenal.multi_view_reconstruction.**reconstruction_3d**(*image_views*, *voxels_size=4*, *error_tolerance=0*, *voxel_center_origin=(0.0, 0.0, 0.0)*, *start_voxel_size=4096*, *voxels_position=None*, *attractor=None*)

Construct a list of voxel represented object with positive value on binary image in images of images_projections.

Parameters

- **image_views** (*[(image, projection), ..]*) – List of tuple (image, projection) where image is a binary image (numpy.ndarray) and function projection (function (x, y, z) -> (x, y)) who take (x, y, z) position on return (x, y) position according space representation of this image
- **voxels_size** (*float, optional*) – Diameter size of the voxels
- **error_tolerance** (*int, optional*) –
- **voxel_center_origin** (*(x, y, z), optional*) – Center position of the first original voxel, who will be split.
- **start_voxel_size** (*int, optional*) – Minimum size that the origin voxel size must include at beginning
- **voxels_position** (*numpy.ndarray, optional*) – List of first original voxel who will be split. If None, a list is create with the voxel_center_origin value.

Returns **out**

Return type *VoxelGrid*

openalea.phenomenal.multi_view_reconstruction.project_voxel_centers_on_image

openalea.phenomenal.multi_view_reconstruction.**project_voxel_centers_on_image**(*voxels_position*, *voxels_size*, *shape_image*, *projection*, *value=255*, *dtype=<type 'numpy.uint8'>*)

Create a image with same shape that shape_image and project each voxel on image and write positive value (255) on it.

Parameters

- **voxels_position** (*numpy.ndarray*) – Voxels center position [[x, y, z], ...]
- **voxels_size** (*float*) – Diameter size of the voxels
- **shape_image** (*2-tuple*) – Size height and length of the image target projected
- **projection** (*function ((x, y, z)) -> (x, y)*) –

  Function of projection who take 1 argument (tuple of position (x, y, z)) and return this position 2D (x, y)
- **value** (*int*) – value between 0 and 255 of positive pixel. By default 255.
- **dtype** (*type*) – numpy type of the returned image. By default numpy.uint8.

Returns **out** – Binary image

Return type numpy.ndarray

openalea.phenomenal.multi_view_reconstruction.project_voxels_position_on_image

openalea.phenomenal.multi_view_reconstruction.**project_voxels_position_on_image**(*voxels_position*, *voxels_size*, *shape_image*, *projection*)

Create a image with same shape that shape_image and project each voxel on image and write positive value (255) on it.

Parameters

- **voxels_position** (*[(x, y, z)]*) – cList (collections.deque) of center position of voxel
- **voxels_size** (*float*) – Size of side geometry of voxel
- **shape_image** (*Tuple*) – size height and length of the image target projected

- **projection** (*function ((x, y, z)) -> (x, y)*) –

  Function of projection who take 1 argument (tuple of position (x, y, z)) and return this position 2D (x, y)

Returns **out** – Binary image

Return type numpy.ndarray

openalea.phenomenal.multi_view_reconstruction.image_error

openalea.phenomenal.multi_view_reconstruction.**image_error**(*img_ref*, *img_src*, *precision=2*)

Return false position and true negative result from the comparaison on two binaries images

openalea.phenomenal.multi_view_reconstruction.reconstruction_error

openalea.phenomenal.multi_view_reconstruction.**reconstruction_error**(*voxels_grid*, *image_views*)

Compute the reconstruction error (false positive and true negative) of the 3d reconstruction from the image view.

Parameters

- **img_ref** (*numpy.ndarray*) – binary image reference
- **projection** (*function ((x, y, z)) -> (x, y)*) –

  Function of projection who take 1 argument (tuple of position (x, y, z)) and return this position 2D (x, y)
- **voxels_position** (*[(x, y, z)]*) – cList (collections.deque) of center position of voxel
- **voxels_size** (*float*) – Size of side geometry of voxel

Returns **out** – Error value

Return type int

Mesh

Algorithms

algorithms.meshing(image_3d[, ...])	Build a polygonal mesh representation (= list of vertices and faces) from a 3d image (= numpy array 3D).
algorithms.marching_cubes(vtk_image_data[, ...])	Call of vtkMarchingCubes on a vtk_image_data with iso_value
algorithms.smoothing(vtk_poly_data[, ...])	Call of vtkWindowedSincPolyDataFilter on a vtk_poly_data to smoothing the edge of poly_data
algorithms.decimation(vtk_poly_data[, ...])	Call of vtkQuadricDecimation on a vtk_poly_data to decimate the mesh

openalea.phenomenal.mesh.algorithms.meshing

openalea.phenomenal.mesh.algorithms.**meshing**(*image_3d*, *smoothing_iteration=0*, *reduction=0.0*, *verbose=False*)

Build a polygonal mesh representation (= list of vertices and faces) from a 3d image (= numpy array 3D).

More, some option, is available to smooth the 3D object representation and reduce the number triangle.

Firstly : A marching cubes algorithm is apply to compute the polygonal mesh.

Secondly : A smoothing algorithm is apply according the number of iteration given

Thirdly : A mesh decimation algorithm is apply according the percentage of reduction given.

Parameters

- **image_3d** (*3D numpy array*) – 3D Array with positive values
- **smoothing_iteration** (*int, optional*) – Number of iteration for smoothing
- **reduction** (*float, optional*) – Center position of the first original voxel, who will be split. 0 and 1
- **verbose** (*bool, optional*) – If True, print for some information of each part of the algorithms

Returns

- **vertices** (*[(x, y, z), ...]*) – Spatial coordinates for unique mesh vertices.
- **faces** (*[(V1, V2, V3), ...]*) – Define triangular faces via referencing vertex indices from vertices. This algorithm specifically outputs triangles, so each face has exactly three indices

openalea.phenomenal.mesh.algorithms.marching_cubes

openalea.phenomenal.mesh.algorithms.**marching_cubes**(*vtk_image_data*, *iso_value=0.5*, *verbose=False*)

Call of vtkMarchingCubes on a vtk_image_data with iso_value

vtkMarchingCubes is a filter that takes as input a volume (e.g., 3D structured point set) and generates on output one or more isosurfaces.

One or more contour values must be specified to generate the isosurfaces. Alternatively, you can specify a min/max scalar range and the number of contours to generate a series of evenly spaced contour values.

Parameters

- **vtk_image_data** (*vtkImageData*) – vtkImageData is a data object that is a concrete implementation of vtkDataSet. vtkImageData represents a geometric structure that is a topological and geometrical regular array of points. Examples include volumes (voxel data) and pixmaps.
- **iso_value** (*float, optional*) – Contour value to search for isosurfaces in volume.
- **verbose** (*bool, optional*) – If True, print for some information of each part of the algorithms

Returns put – vtkPolyData is a data object that is a concrete implementation of vtkDataSet. vtkPolyData represents a geometric structure consisting of vertices, lines, polygons, and/or triangle strips. Point and cell attribute values (e.g., scalars, vectors, etc.)

Return type vtkPolyData

openalea.phenomenal.mesh.algorithms.smoothing

openalea.phenomenal.mesh.algorithms.**smoothing**(*vtk_poly_data*, *feature_angle=120.0*, *number_of_iteration=5*, *pass_band=0.01*, *verbose=False*)

Call of vtkWindowedSincPolyDataFilter on a vtk_poly_data to smoothing the edge of poly_data

Parameters

- **vtk_poly_data** (*vtkPolyData*) – vtkPolyData is a data object that is a concrete implementation of vtkDataSet. vtkPolyData represents a geometric structure consisting of vertices, lines, polygons, and/or triangle strips. Point and cell attribute values (e.g., scalars, vectors, etc.) also are represented.
- **feature_angle** (*float, optional*) – Feature angle for sharp edge identification.
- **number_of_iteration** (*float, optional*) – Number of iteration of smoothing
- **pass_band** (*float, optional*) – Passband value for the windowed sinc filter
- **verbose** (*bool, optional*) – If True, print for some information of each part of the algorithms

Returns **out** – vtkPolyData is a data object that is a concrete implementation of vtkDataSet. vtkPolyData represents a geometric structure consisting of vertices, lines, polygons, and/or triangle strips. Point and cell attribute values (e.g., scalars, vectors, etc.)

Return type vtkPolyData

openalea.phenomenal.mesh.algorithms.decimation

openalea.phenomenal.mesh.algorithms.**decimation**(*vtk_poly_data*, *reduction=0.95*, *verbose=False*)

Call of vtkQuadricDecimation on a vtk_poly_data to decimate the mesh

Parameters

- **vtk_poly_data** (*vtkPolyData*) – vtkPolyData is a data object that is a concrete implementation of vtkDataSet. vtkPolyData represents a geometric structure consisting of vertices, lines, polygons, and/or triangle strips. Point and cell attribute values (e.g., scalars, vectors, etc.) also are represented.
- **reduction** (*float, optional*) – Percentage of reduction for the decimation 0.95 will reduce the vtk_poly_date of 95%
- **verbose** (*bool, optional*) – If True, print for some information of each part of the algorithms

Returns **out** – vtkPolyData is a data object that is a concrete implementation of vtkDataSet. vtkPolyData represents a geometric structure consisting of vertices, lines, polygons, and/or triangle strips. Point and cell attribute values (e.g., scalars, vectors, etc.)

Return type vtkPolyData

Formats

write_vertices_faces_to_ply_file(filename, ...)	Write methods to save vertices and faces in ply format.
write_vtk_poly_data_to_ply_file(filename, ...)	
write_vertices_faces_to_json_file	
read_json_file_to_vertices_faces	

openalea.phenomenal.mesh.write_vertices_faces_to_ply_file

openalea.phenomenal.mesh.**write_vertices_faces_to_ply_file**(*filename*, *vertices*, *faces*, *vertices_colors=None*, *faces_colors=None*)

Write methods to save vertices and faces in ply format.

openalea.phenomenal.mesh.write_vtk_poly_data_to_ply_file

openalea.phenomenal.mesh.**write_vtk_poly_data_to_ply_file**(*filename*, *vtk_poly_data*)

Routines

normals(vertices, faces)	Compute normal of each faces
centers(vertices, faces)	Compute center of each faces

openalea.phenomenal.mesh.normals

openalea.phenomenal.mesh.**normals**(*vertices*, *faces*)

Compute normal of each faces

Parameters

- **vertices** (*[(x, y, z), ..]*) – Spatial coordinates for unique mesh vertices.
- **faces** (*[(V1, V2, V3), ..]*) – Define triangular faces via referencing vertex indices from vertices. This algorithm specifically outputs triangles, so each face has exactly three indices

Returns **out** – List of vector direction of the normal in the same order that faces

Return type [(x, y, z), ..]

openalea.phenomenal.mesh.centers

openalea.phenomenal.mesh.**centers**(*vertices*, *faces*)

Compute center of each faces

Parameters

- **vertices** (*[(x, y, z), ..]*) – Spatial coordinates for unique mesh vertices.

- **faces** (*[(V1, V2, V3), ..]*) – Define triangular faces via referencing vertex indices from vertices. This algorithm specifically outputs triangles, so each face has exactly three indices

Returns **out** – List of center of faces in the same order that faces

Return type [(x, y, z), ..]

VTK Transformation

from_vertices_faces_to_vtk_poly_data(...[, ...])	
from_vtk_poly_data_to_vertices_faces(...)	
from_voxel_centers_to_vtk_image_data(...)	
from_numpy_matrix_to_vtk_image_data(data_matrix)	

openalea.phenomenal.mesh.from_vertices_faces_to_vtk_poly_data

openalea.phenomenal.mesh.**from_vertices_faces_to_vtk_poly_data**(*vertices*, *faces*, *vertices_colors=None*, *faces_colors=None*)

openalea.phenomenal.mesh.from_vtk_poly_data_to_vertices_faces

openalea.phenomenal.mesh.**from_vtk_poly_data_to_vertices_faces**(*vtk_poly_data*)

openalea.phenomenal.mesh.from_voxel_centers_to_vtk_image_data

openalea.phenomenal.mesh.**from_voxel_centers_to_vtk_image_data**(*voxel_centers*, *voxel_size*)

openalea.phenomenal.mesh.from_numpy_matrix_to_vtk_image_data

openalea.phenomenal.mesh.**from_numpy_matrix_to_vtk_image_data**(*data_matrix*)

3D Segmentation

Skeletonization

connect_all_node_with_nearest_neighbors(graph)	Connect all the nodes in the graph together
create_graph(voxels_position, voxels_size)	Create a networkx.graph from voxels positions and voxels_size
graph_from_voxel_grid(voxel_grid[, ...])	Return a weigthed networkx graph builded from a voxel_grid object where each node of the graph is the tuple position of the voxels center.

Continued on next page

Table 36 – continued from previous page

skeletonize(voxel_grid, graph[, subgraph, ...])	Compute phenomenal skeletonization on the voxel_grid based on the graph.
segment_reduction(voxel_skeleton, ...[, ...])	Reduce the number of segments in a VoxelSkeleton object, according to their projection results.

openalea.phenomenal.segmentation.connect_all_node_with_nearest_neighbors

openalea.phenomenal.segmentation.**connect_all_node_with_nearest_neighbors**(*graph*)

Connect all the nodes in the graph together

Parameters **graph** (*networkx.Graph*) –

Returns **graph**

Return type networkx.Graph

openalea.phenomenal.segmentation.create_graph

openalea.phenomenal.segmentation.**create_graph**(*voxels_position*, *voxels_size*)

Create a networkx.graph from voxels positions and voxels_size

Parameters

- **voxels_position** (*list*) – list of 3-tuple
- **voxels_size** (*int*) – Diameter size of voxels

Returns **graph**

Return type networkx.Graph

openalea.phenomenal.segmentation.graph_from_voxel_grid

openalea.phenomenal.segmentation.**graph_from_voxel_grid**(*voxel_grid*, *connect_all_point=True*)

Return a weigthed networkx graph builded from a voxel_grid object where each node of the graph is the tuple position of the voxels center. Each node are edged, if present, to the nodes depict their 26-neigbors in the voxel_grid. The weigth of each edge is the distance between their voxel center position.

If connect_all_point is False, then the graph returned is the subgraph with the biggest connected components. If connect_all_point is True, the subgraph of connected components are edged via the closest neighbors between the subgraph with a weigth equal to the distance between their position.

Parameters

- **voxel_grid** (VoxelGrid) –
- **connect_all_point** (*bool, optional*) –

Returns **graph**

Return type networkx.Graph

openalea.phenomenal.segmentation.skeletonize

openalea.phenomenal.segmentation.**skeletonize**(*voxel_grid*, *graph*, *subgraph=None*, *voxels_position_remain=None*, *mode='plane'*, *plane_width=None*, *ball_radius=None*, *neighbor_size=45*)

Compute phenomenal skeletonization on the voxel_grid based on the graph.

Parameters

- **voxel_grid** (VoxelGrid) –
- **graph** (*networkx.Graph*) –
- **subgraph** (*networkx.graph, optional*) – If not None, perfom the computation of the shorted paths on the subgraph and remove voxels
- **mode** (*str, optional*) – Mode for intercept point along the paths. Two mode available, "ball" or "plane". By default "plane" mode.
- **plane_width** (*int, optional*) – Size in mm of the width of the plane. By default or if None is equal to the voxel_size of the voxel_grid
- **ball_radius** (*int, optional*) – Size in mm of the radius of the ball. By default or if None is equal to the voxel_size * 4 of the voxel_grid

Returns voxel_skeleton

Return type *VoxelSkeleton*

openalea.phenomenal.segmentation.segment_reduction

openalea.phenomenal.segmentation.**segment_reduction**(*voxel_skeleton*, *image_projection*, *required_visible=4*, *nb_min_pixel=100*)

Reduce the number of segments in a VoxelSkeleton object, according to their projection results. Each segments are kept if their projection on the images are not cover by the projection of the other segments in number required_visible. Segments are not cover if their remaining projected pixel are superio to nb_min_pixel.

Parameters

- **voxel_skeleton** (VoxelSkeleton) –
- **image_projection** (*list of tuple (image, projection)*) –
- **required_visible** (*int, optional*) – Number of required not_covered segment to kept it.
- **nb_min_pixel** (*int, optional*) – Number of remaining pixel required to consider segment not covered

Maize Segmentation

maize_segmentation(voxel_skeleton, graph)	Labeling segments in voxel_skeleton into 4 label.
maize_analysis(maize_segmented)	Update info fiel of the VoxelSegmentation object with the analysis result computed.

openalea.phenomenal.segmentation.maize_segmentation

openalea.phenomenal.segmentation.**maize_segmentation**(*voxel_skeleton*, *graph*)

Labeling segments in voxel_skeleton into 4 label. The label are “stem”, “growing leaf”, “mature_leaf”, “unknown”. :param voxel_skeleton: :type voxel_skeleton: openalea.phenomenal.object.VoxelSkeleton :param graph: :type graph: networkx.Graph

Returns **vms**

Return type *VoxelSegmentation*

openalea.phenomenal.segmentation.maize_analysis

openalea.phenomenal.segmentation.**maize_analysis**(*maize_segmented*)

Update info fiel of the VoxelSegmentation object with the analysis result computed. Each organ are a specific algorithm to extract information.

Parameters **maize_segmented** (VoxelSegmentation) –

Returns **maize_segmented**

Return type *VoxelSegmentation*

CHAPTER 4

Authors

- Artzet Simon (simon.artzet@gmail.com)
- Fournier Christian (christian.fournier@inra.fr)
- Christophe Pradal
- Brichet Nicolas (brichet@supagro.inra.fr)
- Chopard Jerome (revesansparole@gmail.com)
- Mielewczik Michael

CHAPTER 5

License

Phenomenal is released under a Cecill-C license.

Note: Cecill-C license is a LGPL compatible license.

Python Module Index

o

Index

Symbols

B

C

D

E

F

W

X

Y

Z