Dynamic model

Oriane Braud

2024-11-04

Table of contents

Preface **3**

1 Introduction **4**

2 Summary **5**

3 Structural cereal model **6**
3.1 Create a parametric leaf 6
3.2 Generate leaf azimuth series 10
3.3 Manage internode lengths 11
3.4 Manage leaf lengths as a function of height 13
3.5 Arrange a leaf to be placed along a stem with a given inclination. 14
3.6 Build the whole plant shoot in 3D, as an MTG. 16
3.7 Display scenes according to different scenarii 19
3.7.1 A single cereal 19
3.7.2 A cereal crop with variability 24
3.7.3 A seemingly growing plant 25
3.7.4 An intercrop organized in rows 27
3.8 Tillering / Branching 28
3.9 Light interception with Caribu 29
3.10 Next steps: 30

4 Dynamic model **31**
4.1 Next steps 37

References **38**

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

2 Summary

In summary, this book has no content whatsoever.

3 Structural cereal model

In this section, we define a static structural model for cereals.
The paragraphs are organized so that it is easy to see how the model is built and how we can play on parameters.

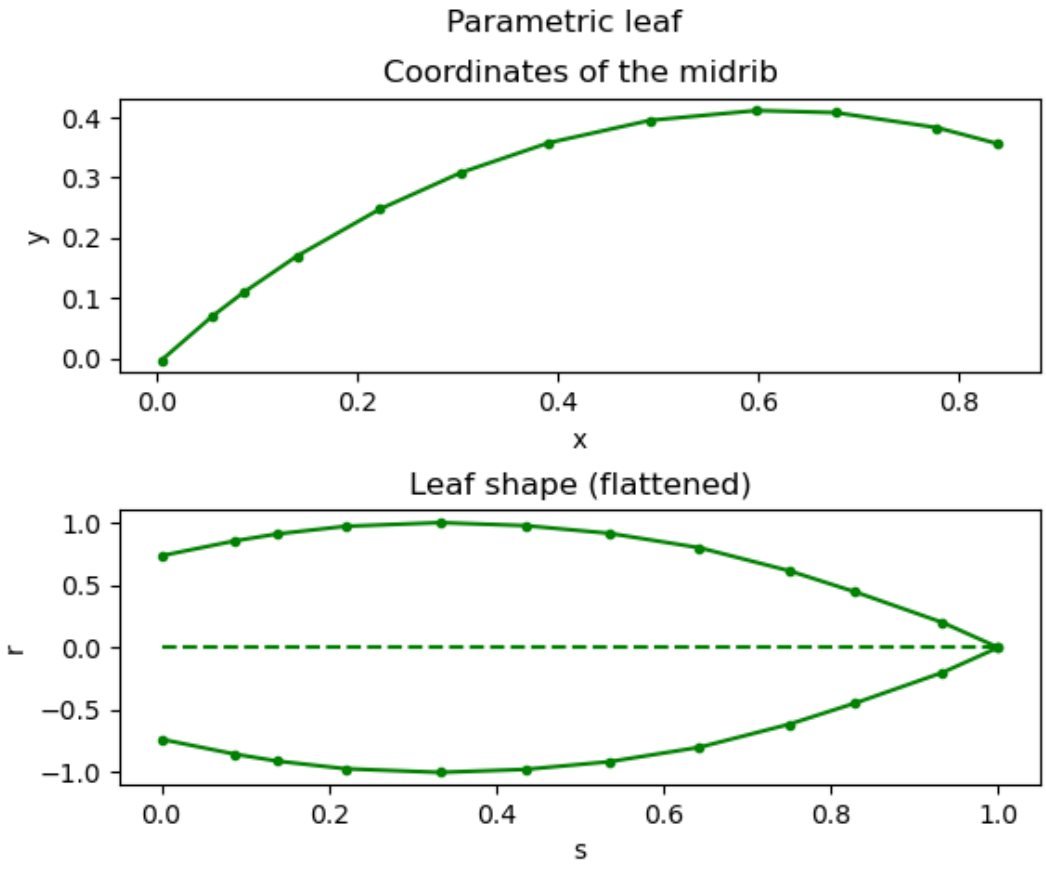
Some 3D plots are interactive, don't hesitate to rotate them.

3.1 Create a parametric leaf

A parametric leaf is here defined by two sets of coordinates:
- (x, y) coordinates for the midrib in a vertical plane (origin = leaf base)
- (s, r) parallel array for curviliear abcissa / relative leaf width along leaf shape

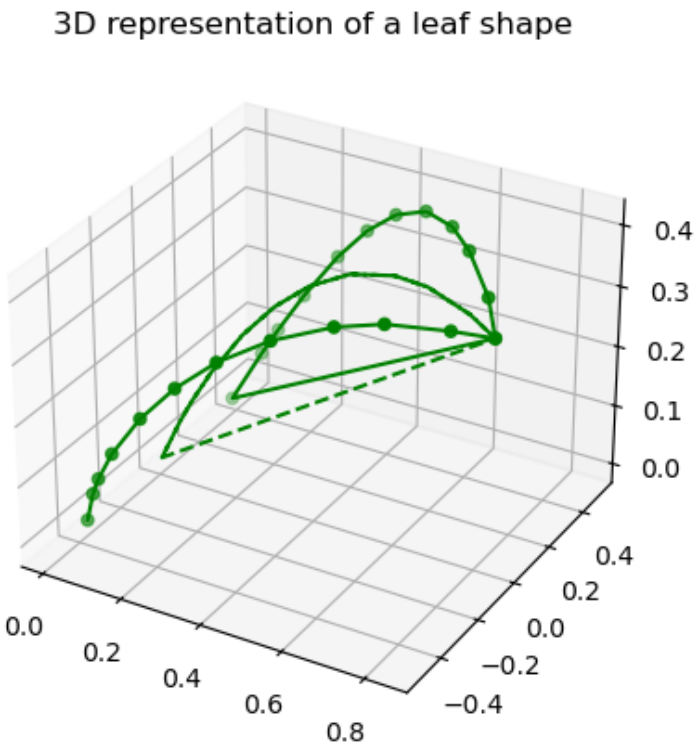
```
## Imports

# from installed packages
import numpy as np
import matplotlib.pyplot as plt
from heapq import *
from scipy.interpolate import splprep, splev
from scipy.integrate import simps, trapz
from openalea.plantgl.all import Vector3

# from ./src
from cereals_leaf import leaf_shape_perez, sr_prevot, parametric_leaf
# or
# from simple_maize import leaf_shape_perez, sr_prevot, parametric_leaf
# from fitting import leaf_shape_perez
from generator import curvilinear_abscisse
from fitting import fit2, fit3, simplify
from simplification import distance, cost

## Code for generating a parametric leaf for a cereal
pl=parametric_leaf(nb_segment=10, insertion_angle=40, scurv=0.7, curvature=70, alpha=-2.3)
```

```
fig, (ax1, ax2) = plt.subplots(nrows=2)
fig.suptitle('Parametric leaf')

ax1.plot(pl[0], pl[1], '.-', c="green")
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title("Coordinates of the midrib")

ax2.plot(pl[2], pl[3], '.-', c="green")
ax2.plot(pl[2], -pl[3], '.-', c="green")
ax2.plot(np.arange(0,1.1,0.1), np.zeros(11), c="green", ls="dashed")
ax2.set_xlabel('s')
ax2.set_ylabel('r')
ax2.set_title("Leaf shape (flattened)")

plt.subplots_adjust(hspace=0.5)

plt.show()
```



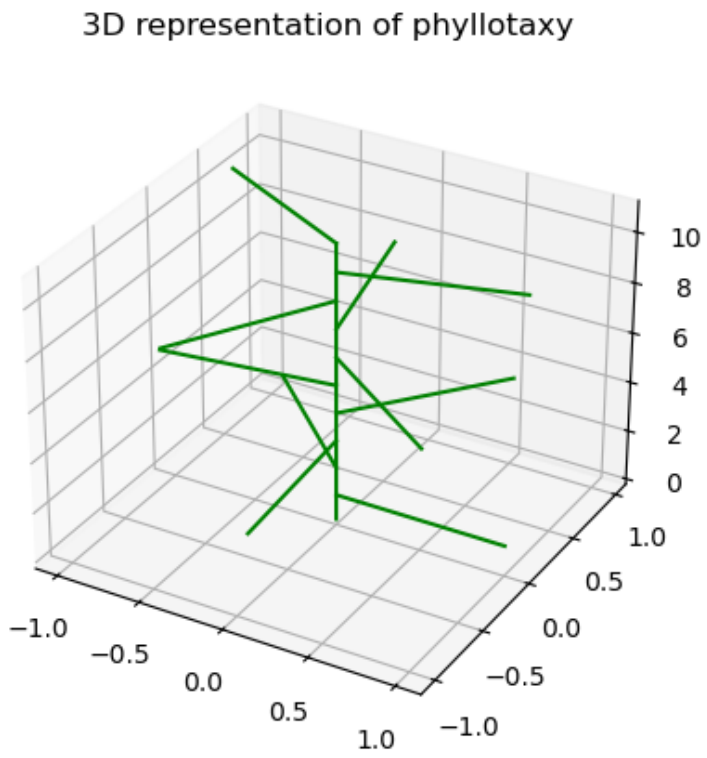

```
## Imports

# from installed packages
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import interp2d
import matplotlib.tri as mtri

# from./src
from fitting import leaf_to_mesh_2d

## Code for representing the parametric leaf in 3D (ignore excess lines)
xy=pl[0:2]
r=pl[3]

pts,ind=leaf_to_mesh_2d(xy[0], xy[1], r)

xs=[pt[0] for pt in pts]
```

```
ys=[pt[1] for pt in pts]
zs=[pt[2] for pt in pts]

X,Y=np.meshgrid(xs, ys)

tri=mtri.Triangulation(xs, ys)

fig=plt.figure()
ax=fig.add_subplot(111, projection='3d')
ax.scatter(xs,ys,zs,c="green")
ax.plot(xs,ys,zs,c="green")
ax.plot(xs,np.zeros(len(ys)),zs,c="green",ls="dashed")
ax.set_title("3D representation of a leaf shape")
plt.show()
```



3.2 Generate leaf azimuth series

```
## Imports

# from installed packages
# from itertools import cycle

# from ./src
from plant_design import leaf_azimuth

## Code for generating leaf azimuths series for a given phyllotaxy
nb_phy=10
phyllotactic_angle=137
spiral=True
phyllotactic_deviation=0
plant_orientation=0

la=leaf_azimuth(size=nb_phy,
                phyllotactic_angle=phyllotactic_angle,
                phyllotactic_deviation=phyllotactic_deviation,
                plant_orientation=plant_orientation,
                spiral=spiral)

x=np.cos(la*np.pi/180)
y=np.sin(la*np.pi/180)
z=np.linspace(1,len(la)+1,len(la))

fig,ax=plt.subplots(subplot_kw=dict(projection='3d'))
for i,a in enumerate(la):
    ax.plot(np.linspace(0,x[i],2), np.linspace(0,y[i],2), [z[i],z[i]], c="green")
ax.plot([0,0], [0,0], [0,z[-1]], c="green")

ax.set_title("3D representation of phyllotaxy")

plt.show()
```

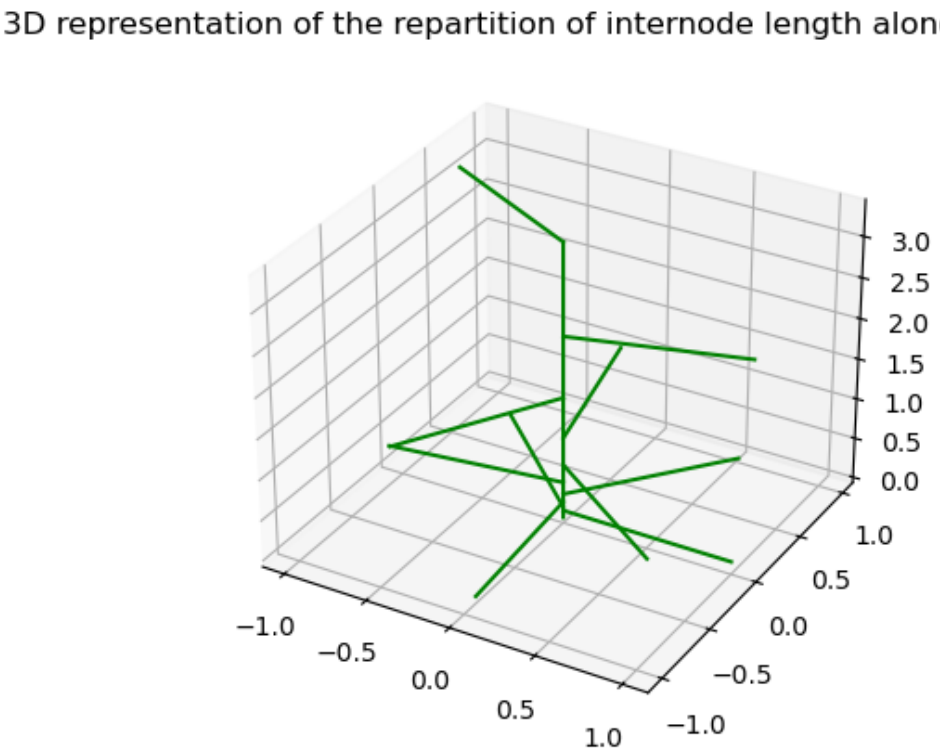


3.3 Manage internode lengths

```
def geometric_dist(height, nb_phy, q=1):
    """ returns distances between individual leaves along a geometric model """

    if q==1:
        u0=float(height)/nb_phy
    else:
        u0=height*(1.-q)/(1.-q**(nb_phy+1))

    return [u0*q**i for i in range(nb_phy)]
```

```
## Code for applying lengths to internodes according to a geometric model
plant_height=15
```

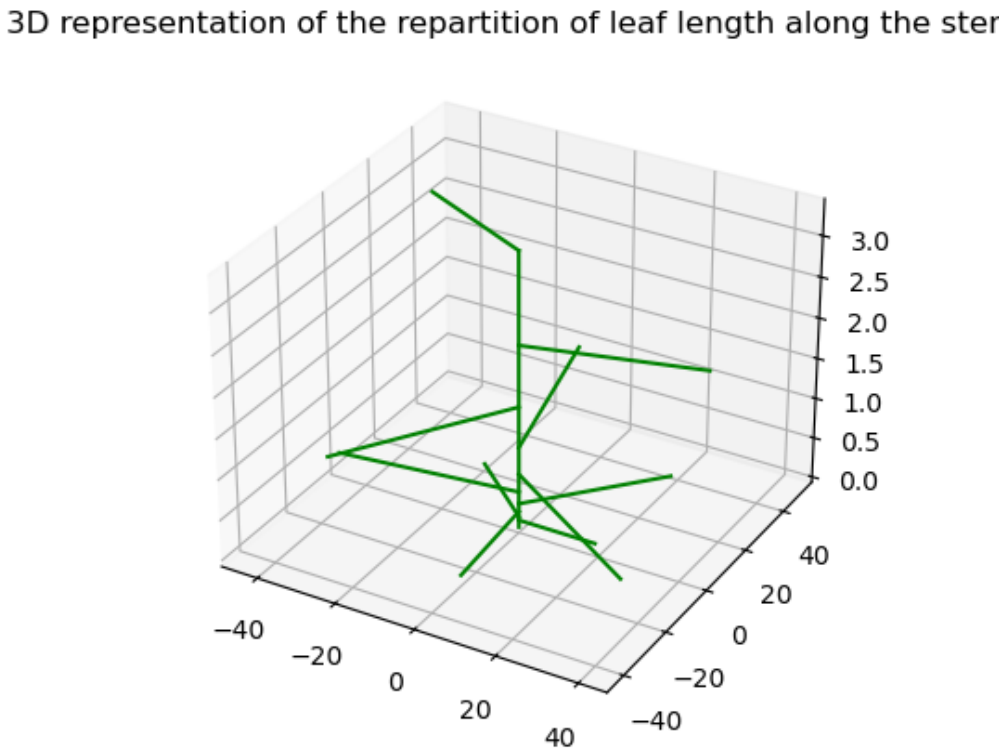
```
q=1.5

x=np.cos(la*np.pi/180)
y=np.sin(la*np.pi/180)
z=geometric_dist(height=plant_height,
                 nb_phy=nb_phy,
                 q=q)

fig,ax=plt.subplots(subplot_kw=dict(projection='3d'))
for i,a in enumerate(la):
    ax.plot(np.linspace(0,x[i],2), np.linspace(0,y[i],2), [z[i],z[i]], c="green")
ax.plot([0,0], [0,0], [0,z[-1]], c="green")

ax.set_title("3D representation of the repartition of internode length along the stem")

plt.show()
```



3.4 Manage leaf lengths as a function of height

```
def bell_shaped_dist(max_leaf_length, nb_phy, rmax=.7, skew=0.15):
    """ returns leaf area of individual leaves along bell shaped model """

    k = -np.log(skew) * rmax
    r = np.linspace(1. / nb_phy, 1, nb_phy)
    relative_length = np.exp(-k / rmax * (2 * (r - rmax) ** 2 + (r - rmax) ** 3))
    # leaf_length = relative_length / relative_length.sum() * max_leaf_length
    leaf_length = relative_length * max_leaf_length
    return leaf_length.tolist()
```

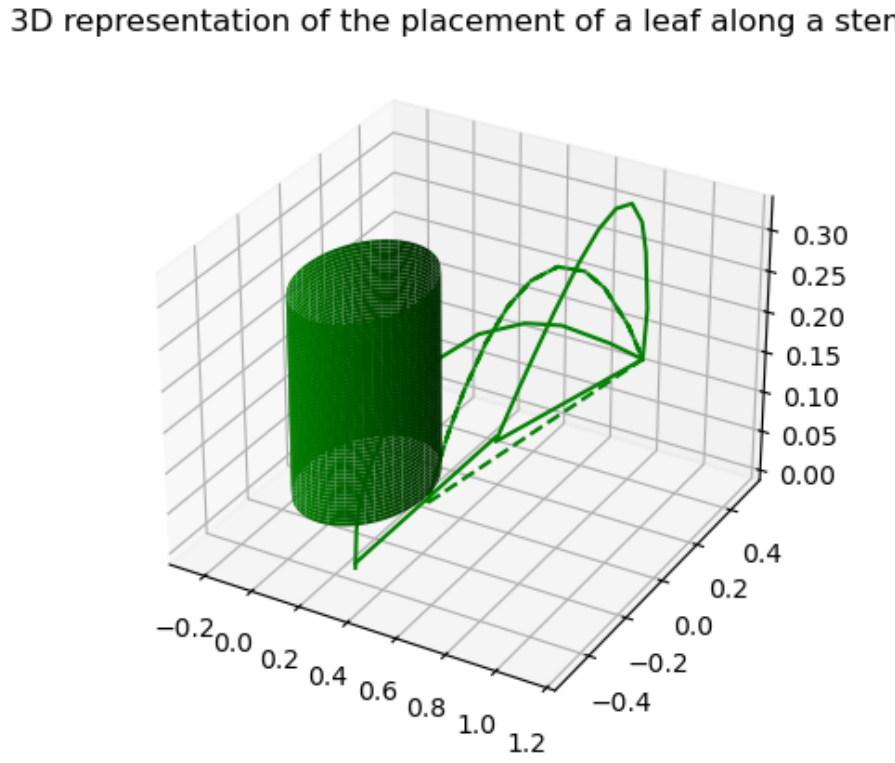
```
## Code for applying lengths to leaves according to a bell shaped model
max_leaf_length=50

bsd=bell_shaped_dist(max_leaf_length=max_leaf_length,
                     nb_phy=nb_phy,
                     rmax=.7,
                     skew=0.15)

x=np.cos(la*np.pi/180)*bsd
y=np.sin(la*np.pi/180)*bsd
z=geometric_dist(height=plant_height,
                   nb_phy=nb_phy,
                   q=q)

fig, ax = plt.subplots(subplot_kw=dict(projection='3d'))
for i,a in enumerate(la):
    ax.plot(np.linspace(0,x[i],2), np.linspace(0,y[i],2), [z[i],z[i]], c="green")
ax.plot([0,0], [0,0], [0,z[-1]], c="green")

ax.set_title("3D representation of the repartition of leaf length along the stem")

plt.show()
```



3.5 Arrange a leaf to be placed along a stem with a given inclination.

```
## Imports

# from installed packages
from math import pi, cos, sin, radians
import openalea.plantgl.all as pgl

# from ./src
# from cereals_leaf import arrange_leaf
# or
from geometry import arrange_leaf
```

```
## Code for placing a leaf against a stem element (here a cylinder), with a given inclinatio
stem_diameter=0.5
inclination=1.2

al=arrange_leaf(leaf=pl,
                stem_diameter=stem_diameter,
                inclination=inclination,
                relative=True)

x=al[0]
y=al[1]
s=al[2]
r=al[3]

pts,ind=leaf_to_mesh_2d(x, y, r)

xs=[pt[0] for pt in pts]
ys=[pt[1] for pt in pts]
zs=[pt[2] for pt in pts]

X,Y=np.meshgrid(xs, ys)

tri=mtri.Triangulation(xs, ys)

fig=plt.figure()
ax=fig.add_subplot(111, projection='3d')
ax.plot(xs,ys,zs,c="green")
ax.plot([xs[0],xs[0]],[ys[0],-ys[0]],[0,0],c="green")
ax.plot(xs,np.zeros(len(ys)),zs,c="green",ls="dashed")

radius=stem_diameter/2
z=np.linspace(0, zs[-1])
theta=np.linspace(0, 2*np.pi)
theta_grid, z_stem=np.meshgrid(theta, z)
x_stem=radius*np.cos(theta_grid)
y_stem=radius*np.sin(theta_grid)

ax.plot_surface(x_stem, y_stem, z_stem, color="green")

ax.set_title("3D representation of the placement of a leaf along a stem")

plt.show()
```



3.6 Build the whole plant shoot in 3D, as an MTG.

```
## Imports

# from installed packages
import openalea.plantgl.all as pgl
from openalea.mtg.turtle import TurtleFrame
from openalea.mtg import MTG, fat_mtg
from scipy.interpolate import interp1d
import pandas

# from ./src
from geometry import slim_cylinder, stem_mesh, _is_iterable, as_tuples, addSets, leaf_mesh, 
# from cereals_leaf import leaf_mesh
```

```
from geometry import CerealsTurtle, CerealsVisitor
from fitting import leaf_element, leaf_to_mesh_2d, leaf_to_mesh, mesh4, plantgl_shape   # lea
from plant_design import get_form_factor, blade_dimension, stem_dimension
from generator import majors_axes_regression, line_projection, as_leaf, as_plant, cereals as


def build_shoot(stem_radius, insertion_heights, leaf_lengths, leaf_areas, leaf_azimuths=None
    """create a shoot

    Args:
        stem_radius: (float) the stem radius
        insertion_heights: list of each leaf insertion height
        leaf_lengths: list of each leaf length (blade length)
        leaf_areas: list of each blade area
        collar_visible: list of each collar height or True if the collar is visible and Fals
        leaf_shapes: list of each leaf shape, if it is not known write None
        leaf_azimuths: list of each leaf azimuth, if it is not known write None

    Returns:
        shoot:

    """
    ranks = range(1, len(leaf_lengths) + 1)
    ntop = max(ranks) - np.array(ranks) + 1
    if leaf_shapes is None:
        a_leaf = parametric_leaf()
        leaf_shapes = [a_leaf for r in ranks]
    if leaf_azimuths is None:
        leaf_azimuths = leaf_azimuth(len(ranks))
    leaf_azimuths[1:] = np.diff(leaf_azimuths)
    ff = [get_form_factor(leaf) for leaf in leaf_shapes]
    blades = blade_dimension(area=leaf_areas, length=leaf_lengths, ntop=ntop)
    stem = stem_dimension(h_ins=insertion_heights, d_internode=np.array(stem_radius) * 2, nto
    df = blades.merge(stem)
    df['leaf_azimuth'] = leaf_azimuths
    df['leaf_rank'] = ranks
    df['leaf_shape'] = [leaf_shapes[n - 1] for n in df.leaf_rank]
    return df, cereals_generator(plant=df)
```

```
def build_shoot_w_pseudo(nb_phy, plant_height, insertion_heights, leaf_lengths, leaf_areas,
                pseudostem_dist=1.4, stem_dist=1.2,
                diam_base=2.5, diam_top=1, pseudostem_height=20,
```

```
            leaf_azimuths=None, leaf_shapes=None, wl=0.1):
    """create a shoot, with pseudostems and stems

    Args:
        stem_radius: (float) the stem radius
        insertion_heights: list of each leaf insertion height
        leaf_lengths: list of each leaf length (blade length)
        leaf_areas: list of each blade area
        collar_visible: list of each collar height or True if the collar is visible and Fals
        leaf_shapes: list of each leaf shape, if it is not known write None
        leaf_azimuths: list of each leaf azimuth, if it is not known write None

    Returns:
        shoot:

    """
    ranks = range(1, len(leaf_lengths) + 1)
    ntop = max(ranks) - np.array(ranks) + 1

    nb_phy = int(nb_phy)

    # Lejeune an Bernier formula + col =
    nb_young_phy = int(round((nb_phy - 1.95) / 1.84 / 1.3))

    # distances between leaves
    pseudostem = geometric_dist(pseudostem_height, nb_young_phy,
                                pseudostem_dist)
    stem = geometric_dist(plant_height - pseudostem_height,
                          nb_phy - nb_young_phy, stem_dist)
    internode = pseudostem + stem
    # stem diameters
    diameter = ([diam_base] * nb_young_phy +
                np.linspace(diam_base, diam_top,
                            nb_phy - nb_young_phy).tolist())

    if leaf_shapes is None:
        a_leaf = parametric_leaf()
        leaf_shapes = [a_leaf for r in ranks]
    if leaf_azimuths is None:
        leaf_azimuths = leaf_azimuth(len(ranks))
    leaf_azimuths[1:] = np.diff(leaf_azimuths)
    ff = [get_form_factor(leaf) for leaf in leaf_shapes]
```

```
    # blades = blade_dimension(area=leaf_areas, length=leaf_lengths, ntop=ntop)
    blades = blade_dimension(length=leaf_lengths, form_factor=ff, ntop=ntop, wl=wl)
    # stem = stem_dimension(h_ins=insertion_heights, d_internode=diameter, ntop=ntop)
    stem = stem_dimension(internode=internode, d_internode=diameter, ntop=ntop)

    df = blades.merge(stem)
    df['leaf_azimuth'] = leaf_azimuths
    df['leaf_rank'] = ranks
    df['leaf_shape'] = [leaf_shapes[n - 1] for n in df.leaf_rank]
    return df, cereals_generator(plant=df)

# shoot, g = build_shoot(3.0, [2,4,6,8], [2,4,6,8], [2,4,6,8])
# scene, nump = build_scene(g)
# display_scene(scene)
```

3.7 Display scenes according to different scenarii

```
## Imports

# from installed packages
from oawidgets.plantgl import *

# Set nice color for plants
nice_green=Color3((50,100,0))
```

3.7.1 A single cereal

```
## Imports

# from installed packages
from oawidgets.plantgl import *

# from ./src
from display import display_mtg, build_scene, display_scene

# Enable plotting with PlantGL
%gui qt
```

```
## Code for generating a 3D cereal shoot from descritive parameters
# Parameters
stem_radius=1
height=1500                 # from crop model
nb_phy=15                   # fixed max nb of phytomers
max_leaf_length=70
insertion_angle=40
scurv=0.7
curvature=80
phyllotactic_angle=120
spiral=True

# Functions calls
insertion_heights=np.array(geometric_dist(height,
                                          nb_phy,
                                          q=1.2)) # further separate stem and pseudo stem, c

leaf_lengths=np.array(bell_shaped_dist(max_leaf_length=max_leaf_length,
                                       nb_phy=nb_phy,
                                       rmax=0.7,
                                       skew=0.15)) # plant area --> max leaf length
# leaf_areas=bell_shaped_dist(plant_area=1, nb_phy=15, rmax=0.7, skew=0.15) # cf blade_dimens

a_leaf = parametric_leaf(nb_segment=10,
                         insertion_angle=insertion_angle,
                         scurv=scurv,
                         curvature=curvature,
                         alpha=-2.3)

leaf_shapes = [a_leaf for l in leaf_lengths] # possible to replace leaf_length by nb_phy or.

leaf_azimuths = leaf_azimuth(size=len(leaf_lengths),
                             phyllotactic_angle=phyllotactic_angle,
                             phyllotactic_deviation=15,
                             plant_orientation=0,
                             spiral=spiral)

shoot, g_single = build_shoot(stem_radius=stem_radius,
                              insertion_heights=insertion_heights,
                              leaf_lengths=leaf_lengths,
                              leaf_areas=None,
                              leaf_shapes=leaf_shapes,
```

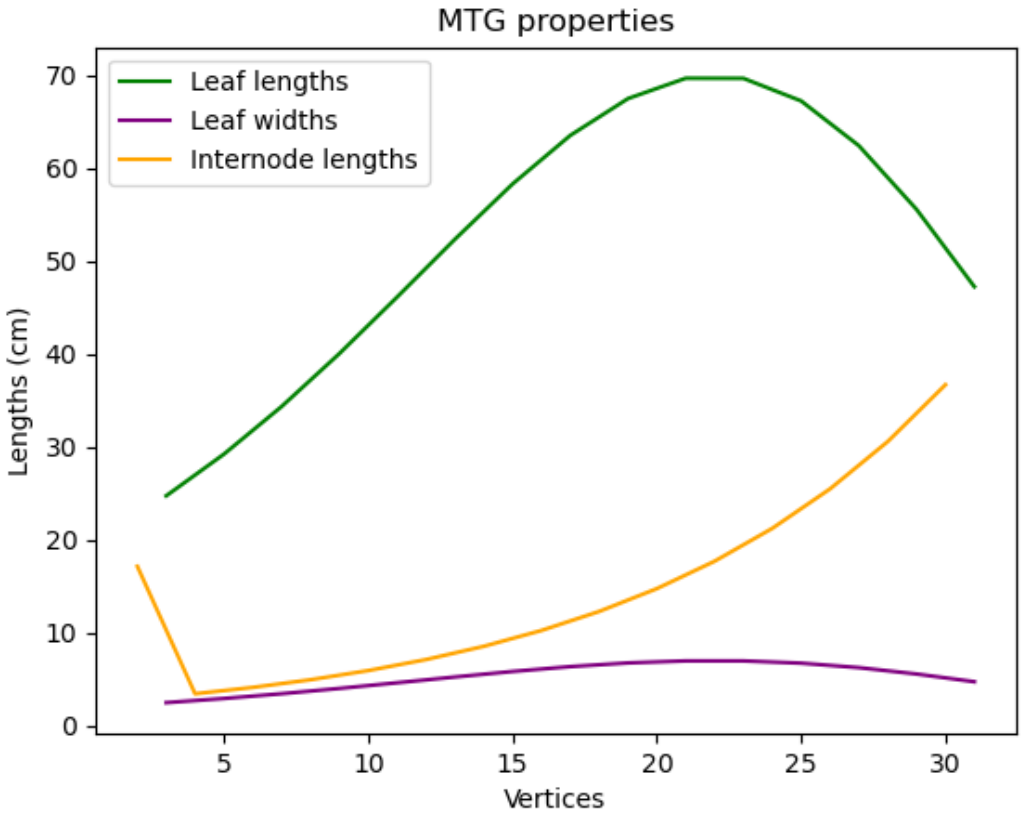
```
                                leaf_azimuths=leaf_azimuths)

# Build and display scene
scene_single, nump = build_scene(g_single,
                                 leaf_material=Material(nice_green),
                                 stem_material=Material(nice_green))
# display_scene(scene_single) # display in separate window
PlantGL(scene_single) # display in notebook
```

```
Plot(antialias=3, axes=['x', 'y', 'z'], axes_helper=1.0, axes_helper_colors=[16711680, 65280
```

```
## Imports

# from installed packages
from oawidgets.mtg import *


## Code for exploring the MTG of the generated cereal shoot

# Properties on the MTG: this exclude all the topological properties
print(g_single.property_names())

# Retrieve one property for the MTG (dict)

labels = g_single.property('label')
# print(labels)

length = g_single.property('length')
# print(length)

leaf_lengths=[]
leaf_ind=[]
internode_lengths=[]
internode_ind=[]
for k,v in length.items():
    if k%2==0: # could have done it using labels
        internode_ind.append(k)
        internode_lengths.append(v)
    else:
        leaf_ind.append(k)
        leaf_lengths.append(v)
```

```
width = g_single.property('shape_max_width')
# print(width)

leaf_widths=[]
for k,v in width.items():
    leaf_widths.append(v)


plt.figure()
plt.plot(leaf_ind, leaf_lengths, c="green", label="Leaf lengths") # == 'shape_mature_length'
plt.plot(leaf_ind, leaf_widths, c="purple", label="Leaf widths")
plt.plot(internode_ind, internode_lengths, c="orange", label="Internode lengths")
plt.xlabel("Vertices")
plt.ylabel("Lengths (cm)")
plt.title("MTG properties")
plt.legend()
plt.show()
```

```
['edge_type', 'label', 'length', 'is_green', 'diameter_base', 'diameter_top', 'azimuth', 'sha
```



The leaf lengths and widths follow the bell shaped curve described before.
The first internode in the MTG actually corresponds to the pseudostem, i.e. the about 4 to 8 short first internodes that rapidly lose their leaves. The lengths following internodes follow the geometric model described before.

```
# classes = list(set(g.class_name(vid) for vid in g.vertices() if g.class_name(vid)))
# print(classes)

# def vertices(g, class_name='P'):
#     return [vid for vid in g.vertices() if g.class_name(vid)==class_name]

# vids_U = vertices(g, 'U')

# plot(g_single, selection=vids_U)

# plot(g_single, selection=[vid for vid in g_single.vertices() if g_single.class_name(vid)==
```

3.7.2 A cereal crop with variability

```
## Imports

# from installed packages
from random import *

# from ./src
from stand import agronomic_plot

# Enable plotting with PlantGL
%gui qt

## Code for generating a random cereal crop from descritive parameters with variability
# Fix a seed
seed(1)

# Initialize the list of plants
plants_in_crop=[]

# Fixed parameters for all plants
length_plot=5
width_plot=5
sowing_density=10
plant_density=10
inter_row=0.5
nplants, positions, domain, domain_area, unit = agronomic_plot(length_plot,
                                                               width_plot,
                                                               sowing_density,
                                                               plant_density,
                                                               inter_row,
                                                               noise=0.1)

# For loop over all the plants in the crop
for n in range(nplants):
    # Parameters varying among plants
    stem_radius=1
    height=1000*(1+random()-0.5)
    nb_phy=15
    max_leaf_length=50*(1+random()-0.5)
    insertion_angle=40*(1+random()-0.5)
    scurv=0.7*(1+random()-0.5)
```

```
    curvature=70*(1+random()-0.5)
    phyllotactic_angle=137*(1+random()-0.5)
    spiral=True

    # Functions calls
    insertion_heights=np.array(geometric_dist(height, nb_phy, q=1.2))
    leaf_lengths=np.array(bell_shaped_dist(max_leaf_length=max_leaf_length, nb_phy=nb_phy, r
    a_leaf = parametric_leaf(nb_segment=10, insertion_angle=insertion_angle, scurv=scurv, cu
    leaf_shapes = [a_leaf for l in leaf_lengths]
    leaf_azimuths = leaf_azimuth(size=len(leaf_lengths), phyllotactic_angle=phyllotactic_angl
                                 plant_orientation=0, spiral=spiral)
    shoot, g_var = build_shoot(stem_radius=stem_radius, insertion_heights=insertion_heights,
                               leaf_areas=None, leaf_shapes=leaf_shapes, leaf_azimuths=leaf_a

    # Fill the list of plants
    plants_in_crop.append(g_var)

# Build and display scene
scene_var, nump = build_scene(plants_in_crop, positions, leaf_material=Material(nice_green),
# display_scene(scene_var)
PlantGL(scene_var)
```

```
Plot(antialias=3, axes=['x', 'y', 'z'], axes_helper=1.0, axes_helper_colors=[16711680, 65280
```

3.7.3 A seemingly growing plant

```
## Inputs

# from installed packages
from openalea.plantgl.all import *

# Enable plotting with PlantGL
%gui qt

## Code for generating a "growing" cereal shoot from descritive parameters
# Fix a seed
seed(1)

# Initialize the list of plant MTGs at different stages
growing_plants=[]
```

```
# Parameters fixing final plant or unchanged with "growth"
nplants=10
positions=[(x,0,0) for x in range(-500, 500, 100)]
final_height=1000
final_nb_phy=2*nplants
heights=geometric_dist(final_height, final_nb_phy, 1.2)
# insertion_heights=np.array(geometric_dist(height, nb_phy, q=1.2))
stem_radius=2
insertion_angle=30
scurv=0.7
curvature=100
phyllotactic_angle=137
spiral=True

# For loop for generating plants at different stages
for n in range(1,nplants+1):
    # Parameters varying when "growing"
    height=10*heights[2*n-1]
    nb_phy=2*n
    max_leaf_length=5*2*n

    # Functions calls
    insertion_heights=np.array(geometric_dist(height, nb_phy, q=1.2))
    leaf_lengths=np.array(bell_shaped_dist(max_leaf_length=max_leaf_length, nb_phy=nb_phy, rn
    a_leaf = parametric_leaf(nb_segment=10, insertion_angle=insertion_angle, scurv=scurv, cu
    leaf_shapes = [a_leaf for l in leaf_lengths]
    leaf_azimuths = leaf_azimuth(size=len(leaf_lengths), phyllotactic_angle=phyllotactic_angl
                                 phyllotactic_deviation=15, plant_orientation=0, spiral=spira
    shoot, g_grow = build_shoot(stem_radius=stem_radius, insertion_heights=insertion_heights
                                leaf_areas=None, leaf_shapes=leaf_shapes, leaf_azimuths=leaf_

    # Fill the list with MTG of "growing" plant
    growing_plants.append(g_grow)

# Build and display scene
scene_grow, nump = build_scene(growing_plants[2:], positions[2:], leaf_material=Material(nice
# display_scene(scene_grow)
PlantGL(scene_grow)
```

```
Plot(antialias=3, axes=['x', 'y', 'z'], axes_helper=1.0, axes_helper_colors=[16711680, 65280
```

3.7.4 An intercrop organized in rows

```
## Inputs

# from installed packages
from openalea.plantgl.all import Material, Color3, Shape, Scene, Viewer, Translated, AxisRota

# Enable plotting with PlantGL
%gui qt

## Code for generating an intercrop from descritive parameters

# Fix a seed
seed(1)

def plant(height, nb_phy, max_leaf_length, phyllotactic_angle, spiral):
    """ return the MTG of a cereal shoot generative from descriptive parameters """
    stem_radius=1
    insertion_angle=30
    scurv=0.7
    curvature=100

    insertion_heights=np.array(geometric_dist(height, nb_phy, q=1.2))
    leaf_lengths=np.array(bell_shaped_dist(max_leaf_length=max_leaf_length, nb_phy=nb_phy, rn
    a_leaf = parametric_leaf(nb_segment=10, insertion_angle=insertion_angle, scurv=scurv, cu
    leaf_shapes = [a_leaf for l in leaf_lengths]
    leaf_azimuths = leaf_azimuth(size=len(leaf_lengths), phyllotactic_angle=phyllotactic_angl
                                 phyllotactic_deviation=15, plant_orientation=0, spiral=spira
    shoot, g = build_shoot(stem_radius=stem_radius, insertion_heights=insertion_heights, lea
                           leaf_areas=None, leaf_shapes=leaf_shapes, leaf_azimuths=leaf_azimu

    return g


# Organize the plant mixture in alternate rows
n_rows = 10
len_rows = 10

d_inter = 70
d_intra = 50

def plant_in_row(i):
```

```
    if i%(4*d_inter)==0 or i%(4*d_inter)==d_inter: return plant(height=1700, nb_phy=15, max_
    else: return plant(height=900, nb_phy=20, max_leaf_length=40, phyllotactic_angle=60, spi


plants_in_intercrop = [plant_in_row(x) for x in range(0, n_rows*d_inter, d_inter) for y in ra

positions=[(x,y,0) for x in range(0, n_rows*d_inter, d_inter) for y in range(0, len_rows*d_in

# Build and display scene
scene_ic, nump = build_scene(plants_in_intercrop, positions, leaf_material=Material(nice_gree
# display_scene(scene_ic)
PlantGL(scene_ic)
```

```
Plot(antialias=3, axes=['x', 'y', 'z'], axes_helper=1.0, axes_helper_colors=[16711680, 65280
```

3.8 Tillering / Branching

In progress ...

```
# Copy MTG single plant
g_branch = g_single.sub_mtg(g_single.root)

# g_branch.display()

# REARRANGE MTG WITH SCALES: PLANT, AXE, METAMER (with internode and leaf), ELEMENTS (StemEle
```

```
# Add branch

from openalea.mtg import MTG

g = MTG()

plant_id = g.add_component(g.root, label='P1')

# scale 1
v1 = g.add_child(plant_id, label='A1')
v2 = g.add_child(plant_id, label='A2')
v3 = g.add_child(plant_id, label='A3')
v4 = g.add_child(v1, label='I1')
v5 = g.add_child(v1, label='I2')
```

```
g.display()

from oawidgets.mtg import *
# plot(g)
```

```
MTG : nb_vertices=7, nb_scales=2
/P1          (id=1)
    /A1          (id=2)
        /I1          (id=5)
        /I2          (id=6)
    /A2          (id=3)
    /A3          (id=4)
```

3.9 Light interception with Caribu

```
## Imports
from alinea.caribu.CaribuScene import CaribuScene
from alinea.caribu.data_samples import data_path

## Code for computing light interception with Caribu
sky = str(data_path('Turtle16soc.light'))
# opts = map(str, [data_path('par.opt'), data_path('nir.opt')])

# complete set of files
cs = CaribuScene(scene=scene_var, light=sky) # opt=opts, ) #pattern=pattern)
raw,agg=cs.run(simplify=True)

# print(raw.keys())

scene,values = cs.plot(raw['Ei'],display=False)

v99 = np.percentile(values, 99)
nvalues=np.array(values)
nvalues[nvalues>v99]=v99
values = nvalues.tolist()

PlantGL(scene, group_by_color=False, property=values)
```

```
Plot(antialias=3, axes=['x', 'y', 'z'], axes_helper=1.0, axes_helper_colors=[16711680, 65280
```

```
## Code for computing light interception with Caribu
sky = str(data_path('Turtle16soc.light'))
# opts = map(str, [data_path('par.opt'), data_path('nir.opt')])

# complete set of files
cs = CaribuScene(scene=scene_ic, light=sky) # opt=opts, ) #pattern=pattern)
raw,agg=cs.run(simplify=True)

scene,values = cs.plot(raw['Ei'],display=False)

v99 = np.percentile(values, 99)
nvalues=np.array(values)
nvalues[nvalues>v99]=v99
values = nvalues.tolist()

PlantGL(scene, group_by_color=False, property=values)
```

```
Plot(antialias=3, axes=['x', 'y', 'z'], axes_helper=1.0, axes_helper_colors=[16711680, 65280
```

3.10 Next steps:

- Configure Caribu to simulate the same light as in STICS
- Adapt for different species (parameters for cereals, some functions for legumes)

4 Dynamic model

```
import sys
# caution: path[0] is reserved for script path (or '' in REPL)
sys.path.insert(1, '../src')
```

We have an MTG which represents the potential plant with a fixed final number of phytomers and branches. At each thermal time step, the turtle visits each element and adds a value to a time series for each (almost) properties of the MTG.

OR

At each thermal time step, the MTG is replicated and the growing elements ar modified, and new elements are added if needed.

Parametrized area of a leaf :

$$\mathcal{S}_normalized = 2|\int_0^1 \mathcal{C}(s(t))ds(t)|$$

$$\mathcal{S}_total = 2 * w * |\int_0^L \mathcal{C}(\frac{s(t)}{L})d\frac{s(t)}{L}|$$

$$\frac{d\mathcal{S}_total}{dt} = 2 * w * |\int_{ds(t)/L} \mathcal{C}(\frac{s(t)}{L})d\frac{s(t)}{L}|$$

- $\mathcal{S}$: leaf area/surface, in cm^2
- $s(t)$: curvilinear abscissa of the midrib, as a function of time, such that:

$$\frac{ds(t)}{dt} = \sqrt{(dx)^2 + (dy)^2}$$

- w : final width of the leaf, in cm
- L : final length of the leaf, in cm
- t : time, in $^oC.day^{-1}$
- $\frac{d\mathcal{S}}{dt}$: gain in surface for a given leaf for a given time step

```
def mtg_turtle_time(g, symbols, time, update_visitor=None ):
    ''' Compute the geometry on each node of the MTG using Turtle geometry.

    Update_visitor is a function called on each node in a pre order (parent before children)
    This function allow to update the parameters and state variables of the vertices.

    :Example:

        >>> def grow(node, time):

    '''

    g.properties()['geometry'] = {}
    g.properties()['_plant_translation'] = {}

    max_scale = g.max_scale()

    def compute_element(n, symbols, time):
        leaf = symbols.get('LeafElement')
        stem = symbols.get('StemElement')

        leaf_rank = int(n.complex().index())
        optical_species = int(n.po)

        metamer = n.complex()

        # Length computation
        if update_visitor:
            length = n.length
            final_length = metamer.final_length
        else:
            final_length = n.final_length
            try :
                length = final_length * (time - metamer.start_tt) / (metamer.end_tt - metamer
            except:
                length = n.length

        if update_visitor and  n.label.startswith('L'):
            if metamer.final_length is None:
                metamer.final_length = n.final_length
                metamer.length = n.length
            length = metamer.length
```

```
            prev_length = metamer.final_length * (n.start_tt - metamer.start_tt) / (metamer.e
            s_base = (metamer.length - prev_length - n.length) / metamer.length
        else:
            s_base = n.srb
        s_top = n.srt
        seed = n.LcIndex
        #leaf inclination

        if update_visitor and  n.label.startswith('L'):
            linc = metamer.insertion_angle
        else:
            linc = n.Linc

        element = {}
        if n.label.startswith('L'):
            radius_max = n.Lw
            element = leaf(optical_species,
                           final_length,
                           length,
                           radius_max,
                           s_base,
                           s_top,
                           leaf_rank, seed, linc)
        else:
            diameter_base = n.parent().diam if (n.parent() and n.parent().diam > 0.) else n.d
            diameter_top = n.diam
            element = stem( optical_species, length, diameter_base, diameter_top)

        can_label =  element.get('label')
        if can_label:
            can_label.elt_id = leaf_rank
            plant_node = n.complex_at_scale(scale=1)
            can_label.plant_id = plant_node.index()

        geom = element.get('geometry')

        return geom, can_label

    def adel_visitor(g, v, turtle, time):
        # 1. retriev the node

        n = g.node(v)
```

```
        # Update visitor to compute or modified the node parameters
        if update_visitor is not None:
            update_visitor(n, time)

            if 'Leaf' in n.label:
                metamer = n.complex()
                if (n.start_tt <= time < n.end_tt) or ((time >= metamer.end_tt) and n.edge_ty
                    angle = float(metamer.Laz) if metamer.Laz else 0.
                    turtle.rollL(angle)
        else:
            if 'Leaf' in n.label:
                if n.edge_type()=='+':
                    angle = float(n.Laz) if n.Laz else 0.
                    turtle.rollL(angle)
            else:
                angle = float(n.Laz) if n.Laz else 0.
                turtle.rollL(angle)

        if g.edge_type(v) == '+':
            angle = n.Ginc or n.Einc
            angle = float(angle) if angle is not None else 0.
            #angle = n.inclination
            #angle = float(angle) if angle is not None else 0.
            turtle.up(angle)

        # 2. Compute the geometric symbol
        mesh, can_label = compute_element(n, symbols, time)
        if mesh:
            n.geometry = transform(turtle, mesh)
            n.can_label = can_label

        # 3. Update the turtle
        turtle.setId(v)

        m = n.complex()
        if update_visitor:
            length = n.length
        else:
            try:
                length = n.length * (time - m.start_tt) / (m.end_tt - m.start_tt) if time < m
            except:
                length = n.length
```

```
        if ('Leaf' not in n.label) and (length > 0.):
            turtle.F(length)
        # Get the azimuth angle


    def traverse_with_turtle_time(g, vid, time, visitor=adel_visitor):
        turtle = PglTurtle()
        def push_turtle(v):
            n = g.node(v)
            #if 'Leaf' in n.label:
                #    return False
            try:
                start_tt = n.complex().start_tt
                if start_tt > time:
                    return False
            except:
                pass
            if g.edge_type(v) == '+':
                turtle.push()
            return True

        def pop_turtle(v):
            n = g.node(v)
            try:
                start_tt = n.complex().start_tt
                if start_tt > time:
                    return False
            except:
                pass
            if g.edge_type(v) == '+':
                turtle.pop()

        if g.node(vid).complex().start_tt <= time:
            visitor(g,vid,turtle,time)
            #turtle.push()
        plant_id = g.complex_at_scale(vid, scale=1)
        for v in pre_order2_with_filter(g, vid, None, push_turtle, pop_turtle):
            if v == vid: continue
            # Done for the leaves
            if g.node(v).complex().start_tt > time:
                print('Do not consider ', v, time)
                continue
```

```
            visitor(g,v,turtle,time)

        scene = turtle.getScene()
        return g

    for plant_id in g.component_roots_at_scale_iter(g.root, scale=max_scale):
        g = traverse_with_turtle_time(g, plant_id, time)
    return g
```

```
def thermal_time(g, phyllochron=110., leaf_duration=1.6, stem_duration=1.6, leaf_falling_rate
    """
    Add dynamic properties on the mtg to simulate developpement
    leaf_duration is the phyllochronic time for a leaf to develop from tip appearance to coll
    stem_duration is the phyllochronic time for a stem to develop
    falling_rate (degrees / phyllochron) is the rate at which leaves fall after colar appeara
    """

    plants = g.vertices(scale=1)
    metamer_scale = g.max_scale()-1

    for plant in plants:
        tt = 0
        v = next(g.component_roots_at_scale_iter(plant, scale=metamer_scale))
        for metamer in pre_order2(g, v):
            end_leaf = tt + phyllochron*leaf_duration
            nm = g.node(metamer)
            nm.start_tt = tt
            nm.end_tt = end_leaf
            nm.frate = leaf_falling_rate / phyllochron
            sectors = [node for node in nm.components() if 'Leaf' in node.label]
            stems = [node for node in nm.components() if 'Stem' in node.label]

            nb_stems = len(stems)
            stem_tt = end_leaf
            dtt = phyllochron*stem_duration / nb_stems
            for stem in stems:
                stem.start_tt = stem_tt
                stem.end_tt = stem_tt+dtt
                stem_tt += dtt

            nb_sectors = max(1,len(sectors))
            sector_tt = end_leaf
```

```
            dtt = phyllochron*leaf_duration/nb_sectors
            for sector in sectors:
                sector.start_tt = sector_tt - dtt
                sector.end_tt = sector_tt
                sector_tt -= dtt

            tt += phyllochron

    return g
```

4.1 Next steps

- appearance and elongation of phytomers (internodes and leaves)
- Constrains from crop model

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.