Greenhouse yield prediction

Niousha Hashemi, Kilian Hunold, Carla Mölbert
Systems Engineering Meets Life Sciences
Project report

Advisor:
Prof. Dr. Visvanathan Ramesh

Prepared at Goethe University Frankfurt am Main
March 8, 2020

Contents

1 Simulation

In order to use the previously proposed models, pictures of tomato plants are required. Since we do not have a greenhouse in which we can take the pictures, a simulation will be used instead. Tomato plants grow according to a pattern. The leaves have a certain shape and the flowers have a certain amount of petals in a certain color. But still each plant is individual, as parameters like the number and position of branches can vary. The simulation should be able to automatically generate a multitude of tomato plants varying in these parameter. Each parameter has a certain range in which it can be selected.
Since we decided to focus the project on the prediction of the amount of leaves on the plant, the simulation of diseases and ripeness becomes secondary. While, we will still explain how these parts could be simulated in general, we will concentrate on the simulation of different amounts of leaves on the plants.

1.1 Building the foundation of the tomato plants

The goal of the simulation is to create a variety of different realistic tomato plants. The first step is therefore to determine which parameters describe a realistic tomato plant. A tomato plant can be split in four major parts; stem, leaves, flowers, fruits. Each of which, have their own properties, which can vary from plant to plant in a certain range. For example, if a tomato plant carries ripe fruits these are red, round and have a certain size. But how ripe the fruits are and the health status of the plant, can lead to variations in size and color of the fruits. And the amount and position of the fruits are variable, as well.
In order to create this variability PlantStudio is used. PlantStudio is a parameter-driven simulation tool, which can be used for the simulation of various non-woody plants throughout their life cycle. It provides various parameters concerning the structure and the growth of the plant. It starts with a collection of questions regarding the optics of the plant, containing information about the stem, leaves, flowers and fruits. This enables an individual adjustment of color, shape, amount and distribution for each of these plant parts. A detailed descriptions of the settings used in this project can be found in table 1. Thanks to the randomize function provided by PlantStudio a multitude of different plants with these parameters can be created.

For each created plant PlantStudio animates a life cycle going from the spearing of the plant to the full-grown tomato plant with ripe fruits. The duration of the life cycle and at which point the fruits start to grow, can be manually adjusted to create a realistic representation.

Table 1: General parameters set in PlantStudio describing the growth behavior of tomato plants.

	Description	Settings
Meristems	Meristems are buds from which leaves and branches grow.	• opposing to each other • medium amount of branches • secondary branches • angle stem/branch: medium
Internodes	Internodes are portions of plant stem between leaves and determine how short or tall, straight or viney, and stiff or flexible the plant will be	• long • medium thickness • introduce some curve to the plant
Leaves	Leaves are drawn using a 3D-object. The leaf is drawn bigger as the plant grows. Leaves are connected to the plant by a stalk called petiole	• pre-build shape: tomato leaf • small size • petiole of medium length • angle stem/leaf: large
Compound leaves	A compound leaf contains >=2 leaflets. Leaflets look like small whole leaves, but fit together in the same pattern for all leaves of the plant. A leaf without leaflets is a simple leaf	• pinnate leaves (seven leaves are ordered feather like) • leaflet are in a medium distance to each other
Inflorescence placement	An inflorescence holds fruits and flowers on a plant. It is divided in apical, which means at the end of the plant stems and axillar, which means between the angles between the stem and the leaf.	• no apical inflorescence • multiple axillary inforescneces (10) • primary stems have a medium length
Inflorescence drawing	Inflorescence can have a variety of shapes	• 10 flowers per inflorescence • placed in spikes • stem thickness: medium
Flowers	Like leaves flowers are drawn as 3D objects, with each 3D object representing one petal of the flower.	• 4 small, yellow pedals • pre-built shape: corn leaves • elongated and pointy
Fruits	Fruits are drawn like flowers, but with a portion of fruit rotated around instead of petals	• common fruit section 1 • 10 fruit sections per fruit • huge size • unripe fruits: green • ripe fruits: red

The created plants have the general parameters of tomato plant but do not look realistic thus far, which can be seen in figure 1. The shortcomings of the simulation, like the missing

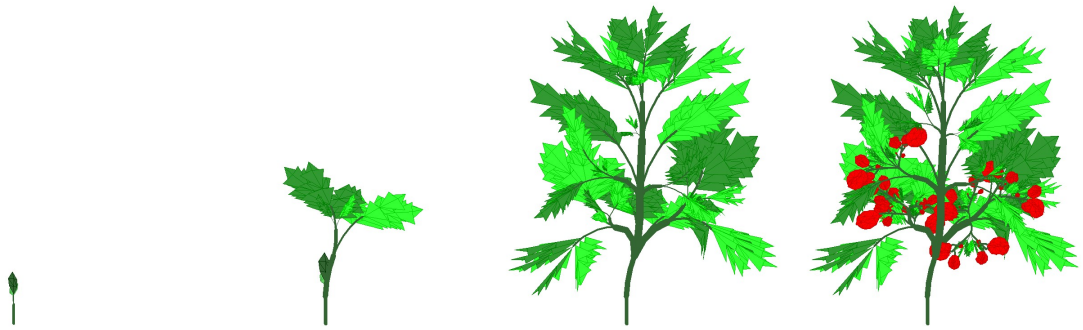

Figure 1: Example of a tomato plant created with PlantStudios in four different stages of growth

texture and material, can be solved by using a second simulation tool, like Blender. This is enabled by PlantStuios option to export the plants as wavefronts (.obj). Here, we created a total of 42 different plants, which were saved grouped by individual plant parts, which allows to change each leaf, fruit or flower individually in later steps.

1.2 Importing and placing plants in Blender

The tomato plant can be imported into Blender. Blender is an open source computer graphics software, which can be used for visual effects, 3D models, etc. The purpose of using Blender is to import the in PlantStudio created plants and add details, shading, rendering and lighting. The script written to import the plants can be easily adjusted in the number of plants that should be planted and the number of plants that are planted in each row. Which plants are planted is randomly selected from the 42 plants exported from PlantStudio.

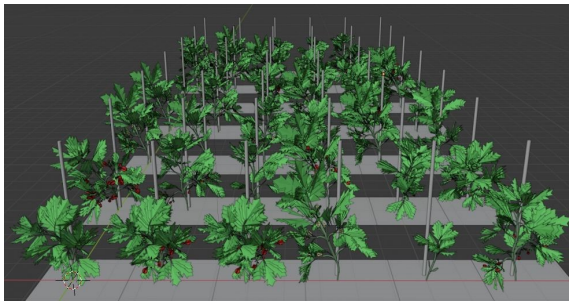

Figure 2: Import of 42 randomly selected plants into Blender with 6 plants per row and soil and pole for each plant.

Listing 1: Given a folder *directory_im* containing obj files a specific number of plants are randomly selected

```
nrPlants = 42
all_files = glob.glob(directory_im + "*.obj")
files = random.choices(all_files, k=nrPlants)
```

The plants are than imported and placed in multiple rows of a specific length, that can be set by the user. Here, it is important to select to split the geometry by group, in order to

individually adjust each leaf or fruit section. In order to be able to differentiate between the various plants, each plant is placed in its own collection.

Listing 2: Inorder to place the plants like they would be placed in the greenhouse we need to know how many plants should be planted and how many fit in one row.

```
    # Place each plant in the scene
    head, tail = os.path.split(f)
    collection_name = tail.replace('.obj', '')
    bpy.ops.import_scene.obj(filepath=f,
     use_split_groups = True, global_clight_size = size)

    # Move each plant in an own collection
    myCol = bpy.data.collections.new(collection_name)
    bpy.context.scene.collection.children.link(myCol)

    # Calculate were to position the plant
    xpos = (counter % plantsRow) * (size / 4)
    ypos = (counter // plantsRow) *  (size / 2)

    # Remove the link to the main context
    for ob in bpy.context.selected_objects:
        myCol.objects.link(ob)
        bpy.context.collection.objects.unlink(ob)
        ob.location.x = xpos
```

Each plant is positioned next to a pole to fix it and on a square of soil. In this step we add these features without any texture and material. These will be added at a later point to further increase the closeness to reality.

Listing 3: Placement of the pole next to each plant. Placement of soil is don analog

```
    length = size / 2.5
    bpy.ops.mesh.primitive_cylinder_add(radius = size / 200,
     depth = length, location = (xpos, ypos + (size / 30),
     length / 2))
    cube = bpy.context.selected_objects[0]
    myCol.objects.link(cube)
    bpy.context.collection.objects.unlink(cube)
```