# THE PLANI PLANT ANIMATION FRAMEWORK

Tina L.M. Derzaph and Howard J. Hamilton

Department of Computer Science, University of Regina, Regina, Saskatchewan

## *ABSTRACT*

*The **PL**ant **ANI**mation (PLANI) framework allows a designer's ideas and decisions about virtual plants to be guided through a structured process that results in an animation of a plant. The process proceeds by selecting relevant objects with properties from four logically grouped domains to simplify implementation. The resulting grouped objects are used as the baseline parameters for the coding process to create the virtual plant. PLANI's construction is based on more than a thousand years of biological research, fifty years of functional-structural plant modelling, and ten years of ontology development, instantiated into an animation environment. PLANI ensures that, when designing virtual plants, a selection of objects derived from an appropriate ontology are considered, and that this selection depends on the purpose of the animation, e.g., whether it is for gaming animation, biological simulation, or film animation. The use of PLANI provides the developer with a framework that is flexible, covers a wide variety of structural, functional, and animation objects for plants, and provides classification of current computer algorithms by their applications to designing virtual plants.*

## *KEYWORDS*

*Ontology, Virtual Plant, Animation, Framework, Plant Animation, Plant Ontology, Gene Ontology, Plant Trait Ontology, Plant Environment Ontology.*

## 1. INTRODUCTION

A plant is a complex object for animation due to the thousands of species and varieties of plants, the hundreds of parts on every plant, and the multitude of calculations required to generate an animation with appropriate motion and growth. This paper proposes an approach to creating virtual plants using a new conceptual framework, called the Plant Animation (PLANI) Framework, which provides a way to use plant ontologies to guide the design and coding of computer algorithms. The PLANI framework includes four domains: FORM (which represents the physiological structure of a plant), FUNCTION (which represents the physiological functioning of a plant), ENVIRONMENT (which represents all environmental influences on a plant), and VIRTUAL WORLD (which represents all relevant computer animation factors for the virtual plant). Where possible, the framework adopts ontologies to organize information about plants and in particular to describe the FORM, FUNCTION, and ENVIRONMENT domains. As a stylistic convention throughout this paper, we give domain names in all uppercase letters (e.g., FORM), object names in mixed case with italics (e.g., *Stem*), and property names in lower case with italics (e.g., *stem length*). Object or property names may consist of multiple words; no hyphens or underscores are added in these cases.

An *ontology* provides semantic classification of entities within a given domain. For example, PLANI uses four plant ontologies [1]: the Plant Ontology (PO) classifies plant anatomy and plant development stages [2, 3, 4], the Gene Ontology (GO) classifies molecular functions, biological processes, and cellular components [5, 6], the Plant Trait Ontology (TO) classify plant traits and characteristics (phenotypes) [7, 8], and the Plant Environment Ontology (EO) classifies the environmental influences on plants [9].

DOI : 10.5121/ijcga.2017.7201 

By guiding the designer through a selection process using plant ontologies to design a virtual plant, PLANI ensures fidelity to biological constraints for simulations while allowing violations of biological constraints for creative plant animation at the discretion of the designer. The remainder of this paper is organized into four sections: Section 2 describes the FORM, FUNCTION, ENVIRONMENT, and VIRTUAL WORLD domains and the design-time and runtime processes; Section 3 describes the implementation and gives examples of how current animations fit into the framework, Section 4 discusses how the PLANI framework can assist in the creation of virtual plants, and Section 5 presents conclusions and suggestions for future work.

## 2. THE PLANI FRAMEWORK

As mentioned in the previous section, the PLANI framework is composed of four domains: FORM, FUNCTION, ENVIRONMENT, and VIRTUAL WORLD, as shown in Figure 1. These four domains enable alignment with the current ontological classifications for plants, including the PO and TO ontologies for FORM, the GO ontology for FUNCTION, and the EO ontology for ENVIRONMENT. By appropriate selection of objects from the four domains, the designer can guide the coder to instantiate virtual plants by using existing algorithms or developing new algorithms. As shown in Figure 1(a), to design a virtual plant using the framework, one proceeds as follows. First, objects in the FORM domain are selected to match the desired structure of the plant. Then, objects in the FUNCTION domain that provide the desired behaviour for the plant are selected. Next, objects in the ENVIRONMENT domain that capture the desired environmental influences on the plant are selected. Lastly, objects in the VIRTUAL WORLD domain that will enable the appropriate animation of the virtual plant using all the selected objects are selected. As an example of the design-time process, suppose the selections in the FORM domain include the *Stem* object, the *Leaf* object, and the *Leaf Colour* object along with their properties. The selections in the FUNCTION domain include the *Growth* object, which contains the *growth rate* property, and the *Response to Nutrient* object, which contains the *limiting nutrient* property. The selections in the ENVIRONMENT domain include the *Seasonal* object, which contains the *Season* property, and the *Nutrient Regimen* object, which contains the *nutrient concentration* property for each nutrient (e.g., boron). The selections in the VIRTUAL WORLD domain include the *World Object* object, which contains the *position* property to describe the placement of the virtual plant in the virtual world.

As part of design, the runtime process should be considered at a high level to ensure that the coding / algorithms will process the objects in the right sequence through the domains. Thus, we will explain the runtime process followed by a coding example. The framework also aids in conceptualizing the runtime process. The processing of the FORM, FUNCTION, and ENVIRONMENT domains proceeds in the opposite order from that used in design, as illustrated in Figure 1(b). The underlying assumption about runtime processing is that objects in the ENVIRONMENT domain influence objects in the FUNCTION domain, which then influence objects in the FORM domain. These influences are implemented through objects in the VIRTUAL WORLD domain, which does not fit neatly into a sequence of processing. At runtime, the objects in the VIRTUAL WORLD domain generate the virtual world, including the *Light* object, the *Viewpoint* object, and the *Topography* object, in which the virtual plant(s) is/are positioned. The runtime process next changes each virtual plant animation by passing property values amongst the corresponding domain objects. It starts by providing values to the ENVIRONMENT domain objects, resulting in setting environment properties, which are then sent to the linked FUNCTION domain objects. The FUNCTION domain objects receive the environment properties and set function properties, which are then sent to the linked FORM domain objects. The FORM domain objects receive function properties and set formproperties based on these function properties. After all properties have been determined, the runtime process displays the state of the virtual plant (one frame of the animation). Repeating the overall process over time gives the virtual plant animation.

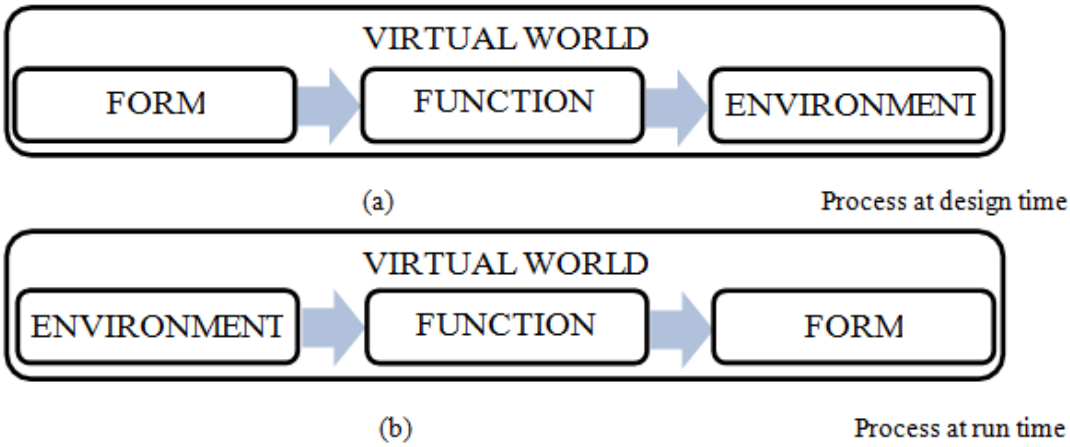


Figure 1: Plant framework domains with process flows

As an example of the runtime process, suppose the *date* property of the *Time* object of the VIRTUAL WORLD is set to August 1, 2014. The *date* property is sent to the *Seasonal* object in the ENVIRONMENT domain. Given this value for the *date* property, the *summer season* property of the *Seasonal* object is set to "Summer". Since the *Seasonal* object in the ENVIRONMENT domain is linked to the *Growth* object in the FUNCTION domain by the *summer season* property, the value of this property is sent to the *Growth* object. The *Growth* object determines the growth rate to be 0.1, given the season value of "Summer". Since the *Growth* object in the FUNCTION domain is linked to the *Leaf* object in the FORM domain via the *growth rate* property, the *growth rate* of 0.1 is sent to the *Leaf* object. Then the *Leaf* object makes any appropriate changes to its properties, such as *leaf length*, *leaf width*, *leaf thickness*, and *leaf elongation rate*.

As another runtime example, suppose the concentration of boron was changed in the soil near the plant, i.e., the *Micronutrient Regimen* object in the ENVIRONMENT domain has the *boron regimen* property changed. The value of this property is sent to the *Response to Nutrient* object in the FUNCTION domain. Then this object sends the *boron regimen* value to the appropriate plant object (such as *Leaf* or *Stem* object) in the FORM domain. This object sets the *colour* property (such as the *leaf colour* or *stem colour* property), which changes the colour of the plant organ based on the *boron regimen* value.

Let us consider the code required to generate the selections in the FORM and VIRTUAL WORLD domains for a simple plant containing one stem and one leaf in a simple virtual world with a designated position for the plant and one light. We assume the structure of the plant is specified by a nested L-system, where an *L-system* (also known as a Lindenmeyer system) is a method of specifying plant structure and function via grammar rules [10]. For simplicity, there is no motion of the plant from the FUNCTION domain and no influence of external factors from the ENVIRONMENT domain in this example. Although it should be understood that in practice existing code will be reused wherever possible, here, for simplicity, we will assume new code is being written. Code is required for: (a) the virtual world, (b) the L- System grammar function, (c) the stem shape, and (d) the leaf shape.

When designing the code for the **VIRTUAL WORLD** domain, it is important to understand that the objects selected from each domain influence how the virtual plant appears.

The code could be written with a starting position for the world model, the plant at the starting position, a viewpoint, and a light aimed roughly at the starting position. For example, at design time, ensuring that the *position* property of the *World Object* object roughly coincides with the

*look at* property of the *Viewpoint* object is essential to viewing the virtual plant. If these two properties do not have similar values, the virtual plant may not be visible to the end-user. In addition, if the *lighting aim direction* property of the *Light* object is not nearly equal to the *position* property, then lighting may not be placed on the virtual plant, which may cause incorrect colouring. Next, code is required for the L-System grammar function. This function is intended to generate three FORM domain objects: *Stem*, *Stem Angle*, and *Leaf*. The code for these objects contains three properties: *axiom*, *production rules*, and *iteration*, which together represent the L-system grammar. The L-system string is generated from the axiom by substituting according to the corresponding production rules. Substitution is continued until the specified number of iterations is reached, resulting in an L-string. Each character in the string represents one of the three objects in the FORM domain, i.e., the *Stem*, *Stem Angle,* or *Leaf*. For example, if the "F" character occurs in the L-string, it represents the *Stem* object; likewise, the "A" character represents a *Leaf* object, and a "(+)" or a "(-)" represents a *Stem Angle* that is positive or negative.

Thirdly, code is required to generate a polygon representing the shape of the stem. More precisely, it generates a *Polygon* object in the VIRTUAL WORLD domain to represent the *Stem* object from the FORM domain. This *Polygon* object has properties for *length*, *width*, and *diameter*, which correspond to the *stem length*, s*tem width*, and *stem diameter* properties of the *Stem* object, respectively, of the FORM domain. The code uses these three properties and the appropriate mathematical formulas to generate a cylinder. For the "F" character example mentioned previously, the stem shape code generates a cylinder of a certain size, representing a *Polygon* object, at run time.

Finally, the leaf shape code generates a *Polygon* object in the VIRTUAL WORLD domain. The *Polygon* object contains properties for *shape, length*, *width*, and *thickness*, which here correspond to the *leaf shape, leaf length*, *leaf width*, and *leaf thickness* properties of the *Leaf* object of the FORM domain. At run time, if an "A" character is present in the L-string, this code generates a polygon object representing a Leaf object. As shown by the examples just discussed, each object and property selected at design time influences the runtime creation and animation of the virtual plant. Since omission of any required object or property from any domain would prevent the construction of the simple virtual plant at runtime, in practise, default values are used throughout to ensure that a complete specification of the virtual plant is produced. The appearance of the plant at runtime provides evidence of the designer's proper selection of objects and the coder's proper implementation.

Clearly, there is a connection between the complexity (the number of objects selected in the various domains) and the amount of coding required for implementation. For example, if the complete *Plant* Ontology in the FORM domain is selected for a single virtual plant, with the objective of realism, more algorithms will be needed than if only a few objects are selected. One can expect an increase in the number of selected objects to cause a corresponding increase in design time, coding complexity, and required computer resources. Overall, the available selections provide a range of possible levels of complexity for the virtual plant. As mentioned, the runtime process is started by the exchange of properties between the four domains, progressing in sequence from ENVIRONMENT to FUNCTION to FORM, all supported by the VIRTUAL WORLD. The specific instantiations of the properties are provided by plant biological observations, whereas the object and property lists are provided through the use of the ontologies.

The designing and coding of plant objects in a virtual world using the PLANI framework is described in the following subsections in detail. The first subsection gives an overview of the VIRTUAL WORLD. The next three subsections describe the selection process in the FORM, FUNCTION, and ENVIRONMENT domains. They also describe some considerations for each of these domains with respect to the VIRTUAL WORLD domain to ensure the virtual plant is

properly visualized. The remaining subsections discuss the coding portion of the simulation and how the elements of the framework combine overall.

## VIRTUAL WORLD

The VIRTUAL WORLD domain consists of the objects used to generate an animation, including *Light*, *Viewpoint*, and *Topology*. This domain enables a virtual plant to be generated for a variety of virtual worlds using the same information from the FORM and FUNCTION domains. Since no formal ontology is currently available for virtual world animation, we adapted a structure from Chu and Li [11]. Their structure contains *WorldObject*, *Transform*, *Translation*, *Rotation*, *Scale*, *Approximation2D*, *Polygon*, *Ground*, *GeometryInfo*, *HotPosition*, *IMWorld* (root object), and *WorldInf* (world information) objects. The *IMWorld* object is at the root of their structure; we replaced this idea by creating the VIRTUAL WORLD domain in our ontology and placing the other objects from their structure in this domain. We renamed *Ground* to *Topology* because the latter term is more descriptive of the many possible topological environments that may be specified in a virtual world, e.g., lakes, hilly terrain, plains, or potted plants in building interiors. Besides the objects considered by Chu and Li, we added objects for *Viewpoint*, *Light*, *Time,* and *Material* because we regard them as relevant to producing animations in a virtual world. *Viewpoint* specifies the position from which the animation is viewed. *Light* specifies the position and direction of a light. *Time* specifies properties for frame rate and, in our case, conversion to real world time for proper virtual plant growth and development. *Material* specifies properties of the surface, including colours (e.g., ambient, diffuse, and specular) and normal vectors. The combination of the viewpoint and the position of the virtual plant in the topology determine the location of the virtual plant in the scene. Lastly, we added the *background colour* property in the *WorldInf* for the scene.

As an example, suppose that at design time the *World Object*, *Viewpoint*, and *Light* objects in the VIRTUAL WORLD domain are selected. The coding assigns values to properties of the selected objects either directly or by passing values from other objects. Further, suppose the *x, y,* and *z* properties of the *position* property are assigned 0, 0, and 0, respectively. At runtime, these values are passed to the *Stem* object in the FORM domain to ensure proper positioning of the virtual plant in the virtual world. As well, these values are passed to the *look at* property of the *Viewpoint* object to ensure that the virtual plant is in focus. Lastly, these values are also passed to the *lighting aim direction* property in the *Light* object to provide the appropriate light on the virtual plant.

## FORM

The FORM domain contains information related to plant morphology, i.e., the shape, texture, and colour of the plant. The ontologies in the FORM domain include the PO and TO ontologies. The PO is the parent of the TO in the plant development class. For example, the leaf sheath (PO:0020104) class, which describes leaf sheathes, has a leaf sheath trait subclass TO:0000835, which adds traits to the leaf sheath, such as diameter (TO:0000642), length (TO:0002689), width (TO:0002721), and color (TO:0002724). So if PO:0020104 is selected there will be less detail than if both PO:0020104 and TO:0000754 are selected. Therefore, when selecting objects, the designer should consider both the PO and the TO objects, depending on the amount of detail required.

Suppose the goal is to produce an animation of a plant with leaves. The designer consults the PO and the TO to select the relevant objects. Figure 2 represents the selection of the *Leaf* object and the *leaf elongation rate*, *leaf width*, *leaf length*, *leaf thickness*, *leaf colour*, and *leaf shape* properties of the FORM domain, as represented by the PO and the TO. The next step is to choose

an implementation approach, either coding or using modelling software. In the first case, the virtual plant is generated procedurally [10; 12]. For example, the leaf shape can be determined by a single formula in the code, given the properties of the leaf [13]. Otherwise, if a modeling tool, such as Maya, is used to generate the form, then a base size is selected to construct the model. Code is used to vary this size at runtime.

For a second example, Figure 3 represents the selection of the *Stem* object and the properties of the object, *stem elongation rate*, *stem length*, *stem width*, *stem colour*, *stem diameter*, *stem strength*, and *stem angle*, represented by the PO and the TO. This selection then needs to be represented by coding to generate the stem, namely the stem function code, which would accept the properties and generate the desired stem structure. The stem function code would generate the *stem shape* property, for example, as a cylinder, and the dimensions of the cylinder would be provided by parameters of the function representing the *stem length*, *stem width*, and *stem diameter* properties.

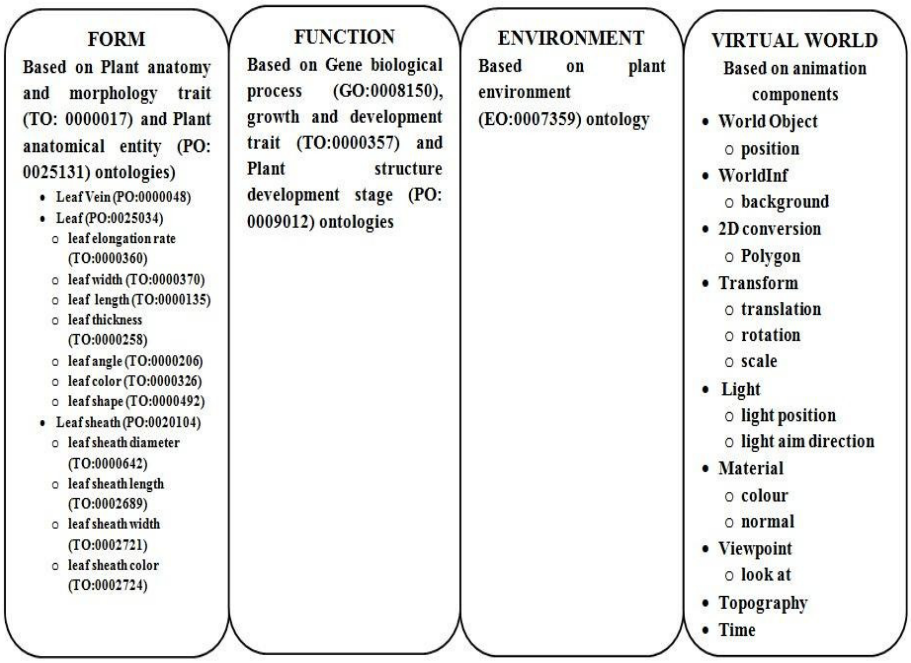


Figure 2: Form selection for the leaf example

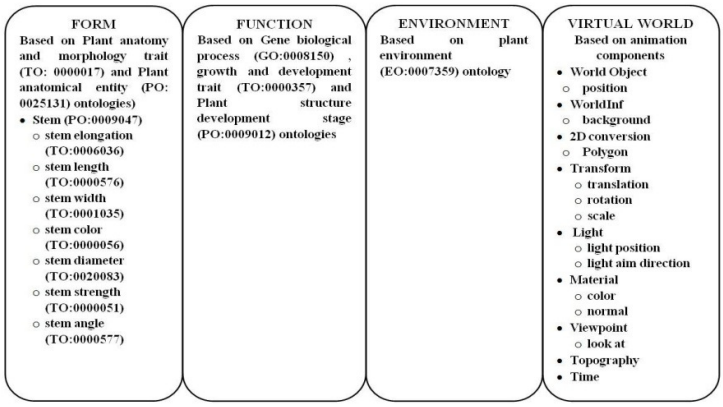


Figure 3: Form selection for the stem example

## FUNCTION

The FUNCTION domain consists of plant motion and plant biological processes such as plant development (e.g., growth, seasonal organ appearance), metabolic processes, and nutrient transport. The ontologies relevant to the FUNCTION domain are the Gene Ontology (GO), plant Trait Ontology (TO) and Plant Ontology (PO). As with the FORM domain, the PO is the parent of the TO, and also is the parent of the GO, with respect to growth and development. For example, the seed development stage (PO:0001170) in the PO is inherited by the TO, where specification with respect to the seed development trait (TO:0000653) provides seed maturation, days to maturity, and germination trait properties that influence seed germination. In addition, the seed development stage is linked to the GO (GO:0048316) through TO inheritance, where the ontology represents the process as the outcome of the seed changes over time through the Growth object.

Figure 4 represents the selection of the *Growth* object from the GO. One property within the *Growth* object is the *growth rate* for a particular plant species. This property is sent to the *Stem* object in the FORM domain, where the property of *stem elongation rate* receives the new *growth rate* and changes the other stem properties accordingly. The selected *Stem* and *Growth* objects would then require coding, where the code parameters represent the properties for each object.

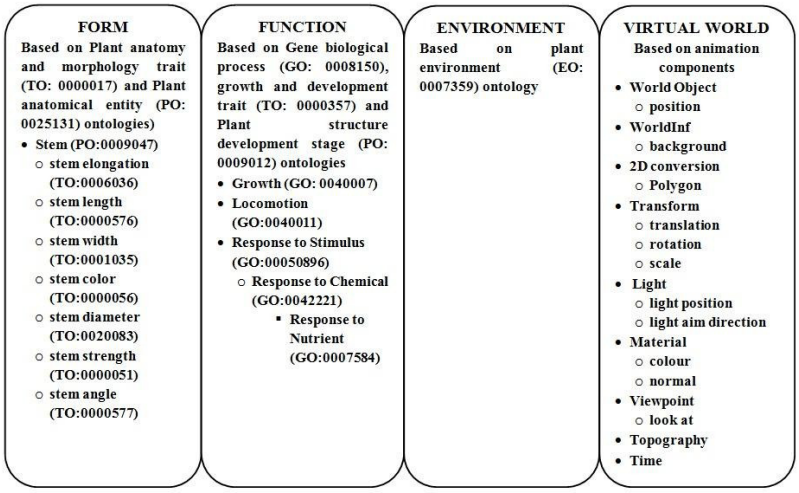


Figure 4: FUNCTION domain selection example

## ENVIRONMENT

The ENVIRONMENT domain consists of the Plant Environment Ontology (EO) [9], as illustrated in Figure 5. The EO includes seasonal changes, radiation regimen (sunlight), temperature, water, gaseous regimen, physical environment (wind, gravity), and chemical regimen (soil nutrients). An environment property, season, for example, affects the FUNCTION domain by activating or deactivating the Growth object, which in turn affects the appearance, size, or nonappearance of the objects in the FORM domain. Another environmental object, Radiation Regimen, is related to the Light object in the VIRTUAL WORLD domain through a brightness property such that a light source has increased/decreased illumination. In addition, the Radiation Regimen object affects the FUNCTION domain's Growth object by increasing or decreasing the growth rate property.

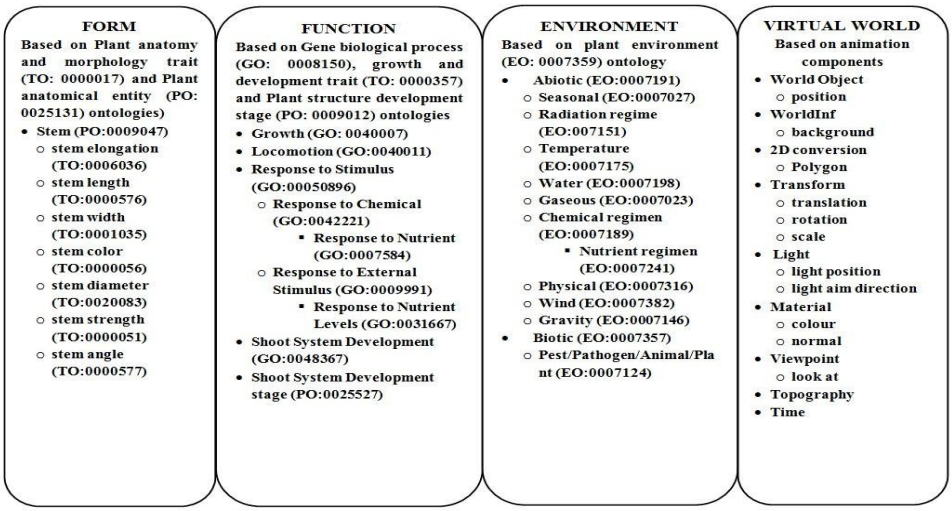


Figure 5: ENVIRONMENT domain selection example

## CODING

It is at the coding step that several considerations need to be incorporated, such as levels of detail and linkage to the other domains. It is recommended that one make the complete object selection across all domains prior to coding any of these objects. This approach allows the opportunity to consider any cross-domain aspects with respect to the properties required to complete the model. For instance, connections between the domains will occur through the coding parameters that represent the object properties. For example, the *Leaf* and *Stem* objects in the FORM domain require the properties of *length*, *width*, *elongation rate*, and *colour*. These objects are represented in the code as the parameters: length, width, elongation rate, and colour. By performing selection from all domains before coding, one ensures that the connection from these objects in the FORM domain to the *Growth* and *Response to Nutrient* objects in the FUNCTION domain is noticed before coding. Advance notice of this relationship ensures that proper coding is performed to smoothly send properties from the FUNCTION domain to the FORM domain at runtime. A detailed example is provided in the Implementation of PLANI section below.

## OVERALL

As mentioned above, the generation of a virtual plant starts with the selection of objects in the FORM, FUNCTION, ENVIRONMENT, and VIRTUAL WORLD domains that provide the most suitable features for the desired purpose. This selection is made according to the ontology for each domain, and is the base activity applied to all domains for design flexibility to enable

gaming animations, biological simulations, or film objects. The framework also provides a way of generating a virtual plant composed of objects to result in a model (FORM domain) and to provide changes to that model through internal (**FUNCTION** domain) or external (ENVIRONMENT domain) parameters in a VIRTUAL WORLD domain. In addition, the framework also gives the overall pattern for runtime; the domains to include, the subsequent properties to include, and the computer algorithm criteria required to fulfill the selections. The extent of design-time selection along with the choice of computer implementation provides the differentiation observed by the viewer.

Let us consider an expanded example that uses the framework for an animation. Figure6 shows the selected domain objects with ontology numbering derived from the current ontological hierarchy [1; 14]. The FORM domain objects include: *Branch* (PO:0025073), *Stem* (PO:0009047), *Petiole* (PO:0020038), and more. There exists an inheritance between the PO and the TO objects, where the TO is more detailed than the PO. The FUNCTION domain objects include: *Growth* (GO:0040007), *Locomotion* (GO:0040011), *Response* to *Stimulus* (GO:0050896), *Shoot System Development* (GO:0048367), *Shoot System Development Stage* (PO:0025527), *Leaf Development* (GO:0048366), *Leaf Development Stage* (PO:0001050), *Bud Development Stage* (PO:0025528), *Fruit Development* (GO:0010154), *Fruit Development Stage* (PO:0001002), *Fruit Ripening* (GO:0009835), *Seed Development Stage* (PO:0001170), *Root Development Stage* (PO:0007520), and *Flower Development Stage* (PO:0007615). Similar to the FORM domain, there is inheritance between the GO and the PO objects in the FUNCTION domain where the GO is the more detailed ontology than the PO. The GO considers changes that occur within the plant, *Topography*, *World Object* (the virtual plant's position in the world), *Time*, and any transformations required for the motion of the plant. Finally, the domain objects can be organized as illustrated in Figure 7.

while the PO considers the external changes that occur to the plant. The ENVIRONMENT domain objects include: *Seasonal* (EO:0007027), *Radiation Regimen* (sun) (EO:0007151), *Temperature* (EO:0007175), *Available Water* (EO:0007198), *Gaseous Regimen* (available gaseous oxygen and carbon dioxide (EO:0007023), *Nutrient Regimen* (available nutrients) (EO:0007241), *Wind* (EO:0007382), *Gravity* (EO:0007146) and *Pathogens* (EO:0007124). Lastly, the objects in the VIRTUAL WORLD domain include: *Light*, *Viewpoint*, 2D *Conversion* of *Polygons*, This additional example was selected to provide illustration of the flexibility of the framework including the additional objects. By dividing the animation of a virtual plant into four domains, the framework enables the designer to plan and select any combination of objects from the online ontologies available. The ontologies provide the base properties that must be coded to simulate a virtual plant. The design time is reduced to a process of searching and selecting rather than starting from scratch. In addition, the designer may gain an understanding of the linkages between the domain objects. Suppose the designer wants to have the leaf move. After the selection of a leaf object in the FORM domain, the notion of movement should cause the designer to notice that the locomotion object for a leaf is required in the FUNCTION domain. Overall, the framework provides implementation flexibility, through the selection of objects and algorithms, and consistency, through the use of four domains that have been formally specified by ontologies. The next section provides an example for how the PLANI framework was utilized in a software implementation.

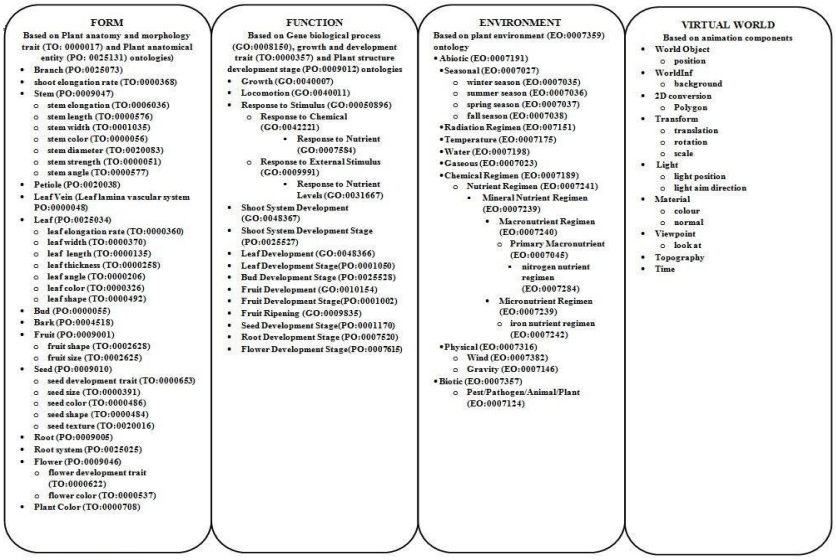


Figure 6: Plant Animation Framework (PLANI) Ontology

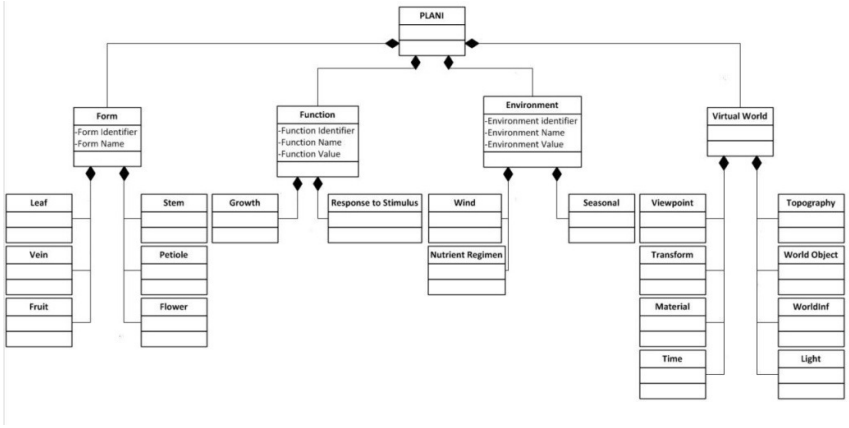


Figure 7: Plant Animation Framework (PLANI) Composition

## 3. IMPLEMENTATION OF PLANI

We implemented a general-purpose software system for creating virtual plants. It supports users who are working in the PLANI framework. As it runs, the system enables the user to select objects from the four domains of the PLANI framework, then generates and displays one plant animation. It provides flexibility by allowing the user to change FUNCTION or ENVIRONMENT objects through data entry or button clicking. The system can generate a wide variety of plant animations, some of which are illustrated in Figure 8.

Of the objects shown in Figure 6, our implementation specifically includes a subset of objects from each domain: the *Stem*, *Leaf*, *Petiole*, *Vein*, and *Fruit* FORM objects, the *Growth, Response to Nutrient, Locomotion, Leaf Development, Fruit Development,* and *Shoot Development* FUNCTION objects, the *Seasonal*, *Wind* and *Nutrient Regimen* ENVIRONMENT objects, and

the *Light*, *World Object*, and *Viewpoint* VIRTUAL WORLD objects. These objects were selected because they are arguably the ones most commonly needed when creating virtual plants for real-time animation, film animation, and biological simulation.

Once the selection of the objects is complete, the implementation approach for the technical instantiation of the virtual plant is considered. We examined three current implementation approaches: procedural generation [15; 16; 17; 18; 19; 20; 21], image capture [22], and modelling. We selected the procedural approach because it provides the greatest flexibility for virtual plant variety with simple coding. In particular, objects in the FORM domain are generated using a nested L-System for each stem, petiole, and vein [23], the *Leaf* object is based on the radius variation formula [24], and the *Fruit* object is generated through the use of an ellipsoid formula. Examples of plants produced by the procedural approach are provided in Figure 8.

Figure 8: Various plant forms utilizing the framework with formulae (original in colour).

The FUNCTION domain algorithms for motion considered the currently available methods for producing the *Locomotion* object. For example, there existed several plant animations for wind that included branches [25; 26; 27; 28], and leaves [22; 27]. In our implementation for the *Stem* object we used a current branch motion algorithm from Sakaguchi and Ohya and added to the algorithm to produce motion in the *Petiole* object [23], and *Leaf* object [24]. In addition to the *Locomotion* object, our implementation required the *Growth* object. Our growth algorithm generated a linear growth with the objective of considering a real- time animation rather than a frame-by-frame animation [23] which was generally based on the growth algorithm created by the Rodkaew et al. paper [30].

Next, consideration of algorithms for the environment; *Wind, Seasonal* and *Nutrient Regimen* objects from the ENVIRONMENT domain are needed. Our implementation uses a simplified wind algorithm to enable real-time animation without demand on the computer resources. The algorithm for the *Seasonal* object uses the passage of daily time where the season is changed when a certain time range is reached; our implementation is based on a simple seasonal assumption of ¼ of a 365.25 day year represents the *spring*, *summer*, *fall*, and *winter season* properties. In addition, an algorithm is required for the *Nutrient Regimen* object along with values for the nutrient ranges, which were collected using various biology research papers. The algorithm implementation determines the *Nutrient Regime* object's specific limiting nutrient, and the limiting nutrient's effect on the subsequent *Growth* objects effects on the s*ize, length, width, diameter*, and *color* properties of the *Stem*, *Leaf*, *Petiole* and *Fruit* objects [31].

The VIRTUAL WORLD in which the virtual plant(s) reside(s) requires the *Light*, *Viewpoint*, *Material*, *Polygon*, *Transformation*, *World Object*, *WorldInf* and *Topography* objects to generate the world. Our implementation uses a consistent light without variation over the entire animation for the implementation of the *Light* object. The *Viewpoint* object's properties enable the viewer to move within and around the VIRTUAL WORLD as is implemented through the x, y, and z position coordinates. The *Polygon* objects used are dependent upon the form algorithms described earlier and each form is located within the virtual world through the *position* property. In Figure 8, the three virtual plants all have a default viewpoint, a default light source, a default position, a default material, and white background from the VIRTUAL WORLD domain.

Lastly, consideration for the interaction between the algorithms is required. This is accomplished through the use of instantiated object properties. For example, in the ENVIRONMENT domain, the *Nutrient Regimen* object contains several *nutrient* properties, one of which is the *nitrogen concentration* property that is part of the *Primary Macronutrient Regimen* object. These properties are passed to the *nutrient* property in the *Response to Nutrient* object of the FUNCTION domain where a process determines the *limiting nutrient* property. This *limiting nutrient* property is passed to the *growth rate property* in the *Growth* object, where it may lead to slower growth. In turn, the *Leaf* object contains the *leaf elongation rate* property, which is changed by the *growth rate* property of the *Growth* object in the FUNCTION domain. Finally, the *Leaf width*, *Leaf length*, and *Leaf thickness* properties are changed by the *leaf elongation rate* property in the FORM domain. For an overview, the FORM, FUNCTION and ENVIRONMENT relationships are shown in Table 1 based on the PLANI sample concepts.

Table 1: Effect of ENVIRONMENT on FUNCTION and FORM

| **FORM Object or property** | **FUNCTION Object** | **ENVIRONMENT Object** |
|---|---|---|
| *Stem* | *Motion* | *Wind* |
| *stem elongation rate* | *Growth* | *Nutrient Regimen* and *Seasonal* |
| *stem colour* | *Response to nutrient* | *Nutrient Regimen* and *Seasonal* |
| *Leaf* | *Motion* | *Wind* |
| *leaf elongation rate* | *Growth* | *Nutrient Regimen* and *Seasonal* |
| *leaf colour* | *Response to nutrient* | *Nutrient Regimen* and *Seasonal* |
| *Fruit* | *Motion* | *Wind* |
| *fruit size* | *Growth* | *Nutrient Regimen* and *Seasonal* |

The coding described in this section was dependent upon the selection of the objects from each of the domains, consideration of the approach for coding (using a modeling software, image capture, or procedural techniques), understanding the relationships between the domain objects, and the selection of the animation type of either gaming, biological simulation, or film. In our PLANI implementation, the general algorithm used to produce the virtual plant is illustrated in Figure 9.

1. Design algorithms for the structure of the plant (based on selections from the FORM domain):
   (a) Design stem using L-System A
   (b) Design petiole using L-System B
   (c) Design vein using L-System C
   (d) Design leaf by selecting the leaf shape and appropriate leaf formula, and appropriate colour.
2. Design algorithms for the functioning of the plant (based on selections from the FUNCTION domain)
   (a) Appearance of form objects (seasonal development)
   (b) Motion effects
   (c) Growth effects
3. Design algorithms for the environmental effects (based on selections from the ENVIRONMENT domain)
   (a) Seasonal
   (b) Wind
   (c) Nutrients

Figure 9: General Algorithm for Sample PLANI implementation

Once the coding is complete, the system software is run to provide the user with a way to work in PLANI. The user is presented with a window that has a variety of selection options, as is illustrated in Figure 10. The Graphical User Interface (GUI) is divided into five sections: Pick a Plant, Make a Plant, Environment of the Plant, Animate Plant, and the resulting virtual plant (at the bottom right of the screen). The Pick a Plant and Make a Plant sections relate to the FORM domain. The Animate Plant section relates to three domains: FUNCTION, ENVIRONMENT, and VIRTUAL WORLD. The Environment of the Plant section relates the ENVIRONMENT domain, and the virtual plant section implements the VIRTUAL WORLD domain. The Make a Plant section provides data entry fields to determine the characteristics of the default virtual plant shown in the bottom right of Figure 10. For example, the *Stem* object properties are the stem length, 1.0, the stem radius is 0.05, and the stem L-String production FBB[+/RFBP][+/RFBP]\\RFBP. The *Petiole* and *Vein* objects are specified similarly. The *Leaf* object contains a drop-down field, with a default of chordate, for the *leaf shape* property. If the user interacts with the software and changes any of the fields in the Make a Plant section, the virtual plant is only altered once the user presses the "Render Make a Plant" button.

As stated earlier, the FUNCTION, ENVIRONMENT, and VIRTUAL WORLD domain objects are represented in the Animate Plant section's buttons and fields. For example, the *Growth* object of the FUNCTION domain is implemented through the "Make it Grow" button, while the *Seasonal* object of the ENVIRONMENT domain is implemented through the "Season On/Off button." In addition, another *Seasonal* object is implemented through the "Add Year" button; each click on this button ages the virtual plant by one year. In contrast, each click on the "Minus Year" button reduces the age by one year. The implementation of the *Viewpoint* object of the VIRTUAL WORLD domain is through the Rotate Left, Rotate Right, Move Back, Move Forward, Rotate Up and Rotate Down buttons. Each button will alter the position of the virtual plan at the bottom right.

Lastly, the ENVIRONMENT domain objects are represented in the Environment of Plant section. The *Plant Creator* software system provides for entry of the concentrations for each of the nutrients for the plant. The virtual plant at the bottom right of Figure 10 is only influenced if the "Make it Grow" button is pushed in the Animate Plant section after the values are entered.
To find a particular species, use the Plant Instance drop down in the Pick a Plant section on the left hand side of the GUI. If the species does not exist, a biologist can use the Plant Creator software to create it, visualize it, and add it to the database with the chosen taxonomic information. Generating a new species of virtual plant is performed by entering values into various fields in the Make a Plant section and visualization is performed by clicking the Render Make a Plant button. This process can be repeated as required until the species is represented properly. Once the biologist is satisfied, the Save New Plant button (middle of the GUI) in the Make a Plant section can be clicked to save the current values into the software's database as shown in Figure 11.

Figure 12 through 14 illustrate several plant forms that can be created and visualized with the Plant Creator software system. Figure 12 shows a tree at year 2 with an oval leaf shape, while Figure 13 shows a tree with a compound palmate leaf. Figure 14 shows a vine with twisting growth. These examples demonstrate the potential of the Plant Creator system software to generate and display a great diversity of plants.

## 4. DISCUSSION

The PLANI framework provides four domains, which enable the designer to link the characteristics of virtual plants to current ontological classifications. The underlying assumption is that objects in the ENVIRONMENT domain influence objects in the FUNCTION domain, which subsequently influence objects in the FORM domain, all of which are supported by the

VIRTUAL WORLD. In this section, we discuss four implementation factors, their relation to the selection of objects in domains, and the overall linkage to plant ontologies. Throughout the discussion, we use three scenarios to provide illustrations.

The three scenarios are shown in Table 2. This table shows selected objects for the sequence of domains from ENVIRONMENT to FORM. For each of the scenarios, PLANI has been or could be used to design a virtual plant. The table provides evidence that PLANI is well suited to a wide variety of applications, regardless of animation type (gaming, biological simulation, or film). The scenarios will be discussed in more detail later in this section.

Table 2: Sample Objects Selected or Three Scenarios

| **Scenario** | **VIRTUAL WORLD objects** | **FORM objects** | **FUNCTION objects** | **ENVIRONMENT objects** |
|---|---|---|---|---|
| **Gaming** | | | | |
| New 3D game creation | *World Object, Position, Polygon, Transformation, Light, Material, Topography* | *Stem, Leaf, Flower* | *Locomotion* | *Radiation Regimen, Wind* |
| **Biological Simulation** | | | | |
| Plant growth affected by nutrients [31] | *World Object, Position, Polygon, Transformation, Light, Material, Topography* | *Stem, Petiole, Vein, Leaf* | *Locomotion, Response to Nutrient, Growth* | *Radiation Regimen, Wind, Nutrient Regimen* |
| **Film** | | | | |
| Animated film with plant | *World Object, Position, Polygon, Transformation, Light, Material, Topography* | *Stem, Leaf, Flower* | *Locomotion* | *Radiation Regimen, Wind* |

The PLANI framework provides flexibility in implementation because a variety of existing algorithms can be chosen for objects in each domain or new algorithms can be added through coding. The design process is four-fold: select one or more domains, one or more objects in the selected domain, one or more properties of the selected objects, and one or more algorithms for the objects in the domains. Once a property has been selected, it is also given a specific value, as needed. The determining factors for guiding or restricting these selections are the implementation style (coding or modelling), the type of animation (gaming, biological simulation, or film), the processing requirements (real-time or batch), and the computer resource capacity (CPU, GPU, and storage).

The first factor, implementation style, can be chosen as one of two types of coding, either L-systems [10] or image capture [22]. These algorithms are directly linked to objects, such as *Stem* and *Petiole,* in the FORM domain. Alternatively, the implementation style could be chosen as 3D modelling, with software such as Maya [32]. All three alternatives represent valid methods to implement plant objects in the FORM domain. Likewise, the FUNCTION domain has implementation style choices either through the use of algorithm coding that moves stems and leaves [28] or through a series of images. Similarly, the ENVIRONMENT domain has implementation style choices between coding and a series of background images. Thus, the choice of implementation style guides the selection of objects for the domains.

The second and third factors are best considered together. Selecting the type of animation will influence the processing requirements and therefore the object selections. Typically, a game requires real-time processing. As a result, the designer should select relatively few objects in the

FORM domain and very few objects from the FUNCTION and ENVIRONMENT domains. In contrast, a film requires batch processing to ensure sufficient processing resources to create a high level of detail. Thus, more objects can be selected from the FORM and FUNCTION domains. However, relatively few objects should be selected in the ENVIRONMENT domain due to the restricted timeframe of the film. A biological simulation can be implemented with either batch or real-time processing, depending on the desired demonstration. The selection of objects from each domain is limited accordingly.

Lastly, the fourth factor, available computing capacity, needs to be considered during design.For instance, many gaming animations use techniques for reduced GPU and CPU usage. One technique is to restrict the representation of the *Leaf* and *Branch* objects in the FORM domain to 2.5D images. A 2.5 image is a collection of planes with textures that are placed on polygons, such as cubes or prisms. The 2.5D images require fewer polygons to be generated than 3D models, but they provide the illusion of 3D plants, as observed in games such as *World of Warcraft*, *Rift*, and *Skyrim*. This technique is appropriate for these games because of the unknown computing capacity and network limitations of the user's machine. In contrast, film requires implementation of complex movements and considerable detail. Generally, the technique employed to increase capacity is to generate detailed objects in the FORM and FUNCTION domains in a frame by frame fashion, which may require using a data center, such as was done for *Avatar* in 2009 [33]. Moreover, a biological simulation typically requires a more detailed animation than a game and a less detailed one than a film. Thus, limitations available capacity may result in restricting the selection of objects in one or two of the domains.

Let us consider the scenarios listed in Table 2 in more detail to illustrate the impact of the four factors on design. For the first scenario, suppose virtual plants are required as background elements in a 3D action-adventure game. Since gaming is the selected animation type, the developer is constrained with respect to CPU and GPU usage and must use real-time processing. Since the plants are being treated as elements in the background that enhance the gaming experience, available processor power is limited. To obtain suitable virtual plants for the game, the developer can restrict the FORM, FUNCTION, and ENVIRONMENT domains in a variety of ways. On example of FORM restriction is to use 2.5D images, as previously discussed. An example of FUNCTION domain restriction is where a constant swaying is included in the animation to simulate wind with little regard to the environment, as has been implemented in several current games, such as *World of Warcraft* or *RIFT*. Moreover, the ENVIRONMENT domain is also often restricted in gaming. A subset of games, including *Skyrim* and *World of Warcraft*, have implemented day/night, cloud, and precipitation variation to enhance the gaming experience, although with little consideration to the effects of these variations on plants. *Skyrim* also provides seasonal plant changes; for example, flowers are present on trees in spring, but not in other seasons. Lastly, since the available computer capacity is unknown, the number of objects selected in the FORM, FUNCTION, and ENVRIONMENT domains need to be further limited.

For the second scenario, suppose we want to animate the process of growth in a detailed biological simulation. To do so, the ENVIRONMENT, FUNCTION, and FORM domains can be implemented in greater detail than in gaming. However, biological simulations often restrict the number of FORM and FUNCTION objects to enable study of a particular effect. For example, to create an animation showing the effect of nutrients on plant growth, one could restrict the FUNCTION domain objects to *Growth* and *Response to Stimulus* [31]. Similarly, to create an animation showing the effect of wind on plant growth, one could restrict the FUNCTION domain objects to *Growth* and *Locomotion* [23]. If sufficient computer capacity to run all three objects of the FUNCTION domain (*Growth*, *Locomotion*, and *Response to Stimulus*), using real-time processing, then restrictions need not occur. As a last example, to study apical meristem growth, Prusinkiewicz et al. restrict each of the FORM and FUNCTION domains to a few objects [34].

They consider the growth object in the FUNCTION domain and its influence on the apical meristem object in the FORM domain.

Finally, for the third scenario, to perform animation for a film, a large number of objects may be selected in the FORM and FUNCTION domain to ensure sufficient detail. As well, the objects selected in the ENVIRONMENT domain can be restricted to the time period of the film. For example, *Leaf Sheaths* may be selected in addition to *Leaf* objects and properties. These sheaths can be shown in a close-up as the film changes focus from the plant to the main characters. In the *Avatar* film, let us consider the scene where the main character touches one plant and all plants rapidly close. To design this scene with PLANI, the objects could be restricted to *Leaf* in the FORM domain and *Locomotion* in the FUNCTION domain. As film is the selected implementation style, the processing could be limited to batch to enable the frame by frame design and creation of the detail required. The GPU and CPU requirements may be high if the images of plants are required to be realistic and aesthetically pleasing. The above three scenarios illustrate how PLANI can be applied to a wide variety of virtual plant requirements. In addition, PLANI aids in organizing and validating the link between the design of a virtual plant and any available plant ontologies regardless of implementation style.

## 5. CONCLUSION

The PLant ANImation (PLANI) framework provides a way to use plant ontologies to guide the design, coding, simulation, and animation of virtual plants. The use of the ontologies when selecting relevant objects allows the design to proceed in a well organized manner while considering biological factors. The key to the approach is the combination of the four interrelated domains (FORM, FUNCTION, ENVIRONMENT and VIRTUAL WORLD) with current plant ontologies. The framework enables a virtual plant designer to consider all aspects of animating a virtual plant by providing all relevant objects in a hierarchical format. It provides a structure for considering inclusion or exclusion based on four stages: (1) selection of required domains, (2) selection of the objects and properties in the chosen domains, (3) selection or design of algorithms for the chosen objects and properties, and (4) coding of the algorithms, as necessary, to produce a virtual plant based on appropriate resource constraints for the application (gaming, biological simulation, or film). In addition, the framework provides a way to directly link each animation object to the corresponding biological object; this linkage promotes consistency between the objects. Regardless of the animation approach (mathematical, image-capture, or modelling tool usage), the designer is guided by the PLANI framework to produce a plant animation in the context of the stated design and implementation considerations.

The framework was implemented in the Plant Creator software system and applied to create a variety of virtual plants. With this system, the designer can select properties of objects in the FORM domain (*Stem*, *Petiole*, *Leaf*, and *Vein*), FUNCTION domain (*Growth* and *Motion*) and ENVIRONMENT domain (*Wind* and *Season*) via the interface. As the designer changes the values of the properties they directly produce animations of the corresponding virtual plants in the VIRTUAL WORLD domain. Examples showed virtual plant default, species specific example *Gingko biloba*, a tree with oval leaves, a tree with compound leaves, and a vine. This software system could be easily expanded by including more objects in its four domains. For example, *Flower* and *Seed* objects could be added to the FORM domain, *Response to Temperature* and *Response to Disease* objects could be added to the FUNCTION domain, *Temperature* and *Disease* could be added to the ENVIRONMENT domain, and more complex lighting and topology objects could be used in the VIRTUAL WORLD domain.

## REFERENCES


[1] Walls, R.L., Athreya, B., Cooper, L., Elser, J., Gandolfo, M.A., Jaiswal, P., Mungall, C.J., Preece, J., Rensing, S., Smith, B., Stevenson, D.W., (2012) Ontologies as integrative tools for plant science, American Journal of Botany, Vol. 99, no.8, pp.1263-1275.

[2] Plant ontology website, (2013) Plant Ontology Consortium, www.plantontology.org, retrieved September 1, 2014.

[3] Avraham, S., Tung, C.-W., Ilic, K., Jaiswal, P., Kellogg, E.A., McCouch, S., et al., (2008) The Plant Ontology Database: a community resource for plant structure and developmental stages controlled vocabulary and annotations, Nucleic Acids Res., Vol. 36, pp. D449–454.

[4] Cooper, L., Walls, R.L., Elser, J., Gandolfo, M. A., Stevenson, D. W., Smith, B., Preece, J., Athreya, B., Mungall, C. J., Rensing, S., Hiss, M., Lang, D., Reski, ., Berardini, T. Z., Donghui, L, Huala, E., Schaeffer, M., Menda, N., Arnaud, E., Shrestha, R., Yamazaki, Y., Jaiswal, P., (2012) The Plant Ontology as a tool for comparative plant anatomy and genomic analyses, Oxford University Press, pp. 62.

[5] GENE Ontology Consortium, (2009) The Gene Ontology in 2010: Extensions and refinements, Nucleic Acids Research, Vol. 38, pp. D331 – D335.

[6] Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J. M., Davis A. P., et. al., (2000) Gene ontology: Tool for the unification of biology. The Gene Ontology Consortium, Nature Genetics, Vol. 25, pp. 25–29.

[7] Jaiswal, P., Ware, D., Ni J., Chang, K., Zhao, W., Schmidt, S., Pan, X., Clark, K., Teytelman, L., Carinhour, S., Stein, L., McCouch, S., (2002) Gramene: development and integration of trait and gene ontologies for rice, Comparative and Functional Genomics, Vol. 3, no. 2, pp. 132-136.

[8] Plant Trait Ontology website, (2014) National Centers for Biomedical Ontology, bioportal.bioontology.org, retrieved September 1, 2014.

[9] Gramene, (2015) Plant Ontology searches, http://archive.gramene.org/plant_ontology/, retrieved March 15, 2015.

[10] Prusinkiewicz, P., (2004) Art and science for life: Designing and growing virtual plants with L- systems, Acta Horticulturae Vol. 630, pp. 15-28.

[11] Chu, Y.-L., Li, T.-Y., (2012) Realizing Semantic Virtual Environments with Ontology and Pluggable Procedures, Applications of Virtual Reality, eds. C. S. Lanyi, InTech, pp. 171-184.

[12] Deussen, O., Colditz, C., Stamminger, M., & Drettakis, G., (2002) Interactive visualization of complex plant ecosystems, In Proceedings of the Conference on Visualization, pp. 219-226.

[13] Gielis, J., (2003) A generic geometric transformation that unifies a wide range of natural and abstract shapes, American Journal of Botany, Vol. 90, no.3, pp. 333-338.

[14] Buskiewich, R., CoeE, H., Jaiswal, P., McCouch, S., Polacco, M., Stein, L., Vincent, L., Ware, D., (2002) The Plant Ontology ™ Consortium and plant ontologies, Comparative and Functional Genomics, Vol. 3, no. 2, pp. 137-142.

[15] 7th International Conference on Functional – Structural Plant Models (FSPM), (2013), 9-14, June 2013, in Hotel Riekonlinnan, Saariselkä, Finland –www.metla.fi/fspm, 2013.

[16] 6th International Workshop on Functional-Structural Plant Models (FSPM), Sept. 12-17, (2010), Davis CA, the USA, Proceedings: Annals of Botany, Vol. 108, no. 6, 2011.

[17] 5th International Workshop on Functional-Structural Plant Models (FSPM), Nov. 4-9, (2007), Napier, New Zealand, Proceedings: Functional Plant Biology, Vol. 35, no. 9/10, 20084

[18] 4th International Workshop on Functional-Structural Plant Models, 7-11 June, (2004), Montpellier, France, Proceedings: New Phytologist, vol. 166, no. 3, 2005.

[19] 3rd International Workshop on Functional-Structural Tree and Stand Models, 27-30 Sept., (2001) Val-Morin, Canada, Sustainable Forest Management Network, pp. 50.

[20] 2nd International Workshop on Functional-Structural Tree Models, 12-14 Oct., (1998), Clermont- Ferrand, France, Proceedings: Annals of Forest Science, Vol. 57, no. 5/6, 2000.

[21] 1st International Workshop on Functional-Structural Tree Models, Helsinki Workshop on Functional-Structural Tree Models 12-13 Dec., (1996), Helsinki, Finland, Proceedings: Silva Fennica, Vol. 31, no. 3, 1997.

[22] Wu, E., Chen, Y., Yan, T., Feng, J., (1999) Reconstruction and physically-based animation of trees from static images, In Eurographics, Computer Animation and Simulation'99, New York: Springer, pp. 57-166.

[23] Derzaph, T. L. M., Hamilton H. J., (2013) Effects of wind on virtual plants in animation, International Journal of Computer Games Technology, article ID 674848, pp. 11.

[24] Derzaph, T.L.M., Hamilton, H.J., (2013) "Parametric methods for generating leaf geometry", Technical Report CS-2013-02, Department of Computer Science, University of Regina, Canada.

[25] Sakaguchi, T, Ohya J., J., (1999) Modeling and animation of botanical trees for interactive virtual environments, Symposium on Virtual Reality Software and Technology (VRST99), pp. 139-146.

[26] Hu, X., Tao, W., Guo, Y., (2008) Using FEM to predict tree motion in a wind field, Journal of Zhejang University Sci. A., Vol. 9, no. 7, pp. 907-915.

[27] Long, J., Reimschussel, C., Britton, O. Jones, M., (2009) Motion capture for natural tree animation, SIGGRAPH, New Orleans, Louisiana.

[28] Yildiz, C., (2010) "An Implementation on Real-time Animation of Trees Swaying in Wind Fields", Computer Engineering Department, Bilkent University. Ankara, Turkey.

[29] Feng, J. Chen, Y, Tan, T., Wu, E., (2001) Leaf Movement Simulation, J. Comput. Sci. and Technol., Vol. 16, no. 2, pp.189-192.
[30] Rodkaew Y, Chuai-Aree S, Siripant S, Lursinsap C, Chonstitvatana P., (2004) Animating plant growth in L-System by parametric functional symbols, International Journal of Intelligent Systems, Vol. 19, no. 1-2, pp. 9-23.
[31] Derzaph, T.L.M, Hamilton, H. J., (2015) Animating plant growth using nutrients, In press.
[32] Maya, (2014) AutoDesk User guide 2014, retrieved May 28, 2015, http://download.autodesk.com/global/docs/maya2014/en_us/index.html?=contextId=BULLETNODESMay28, 2015
[33] Rath, John. (2010) Avatar, Hollywood and the Data Center, Data Center Knowledge, www.datacenterknowledge.com.
[34] Prusinkiewicz, P., Palubicki, W., (2010) What determines tree form?, Proceedings of the 6th International Conference on Functional-Structural Plant Models, Annals of Botany, Vol. 108, no. 6, pp 2-5.

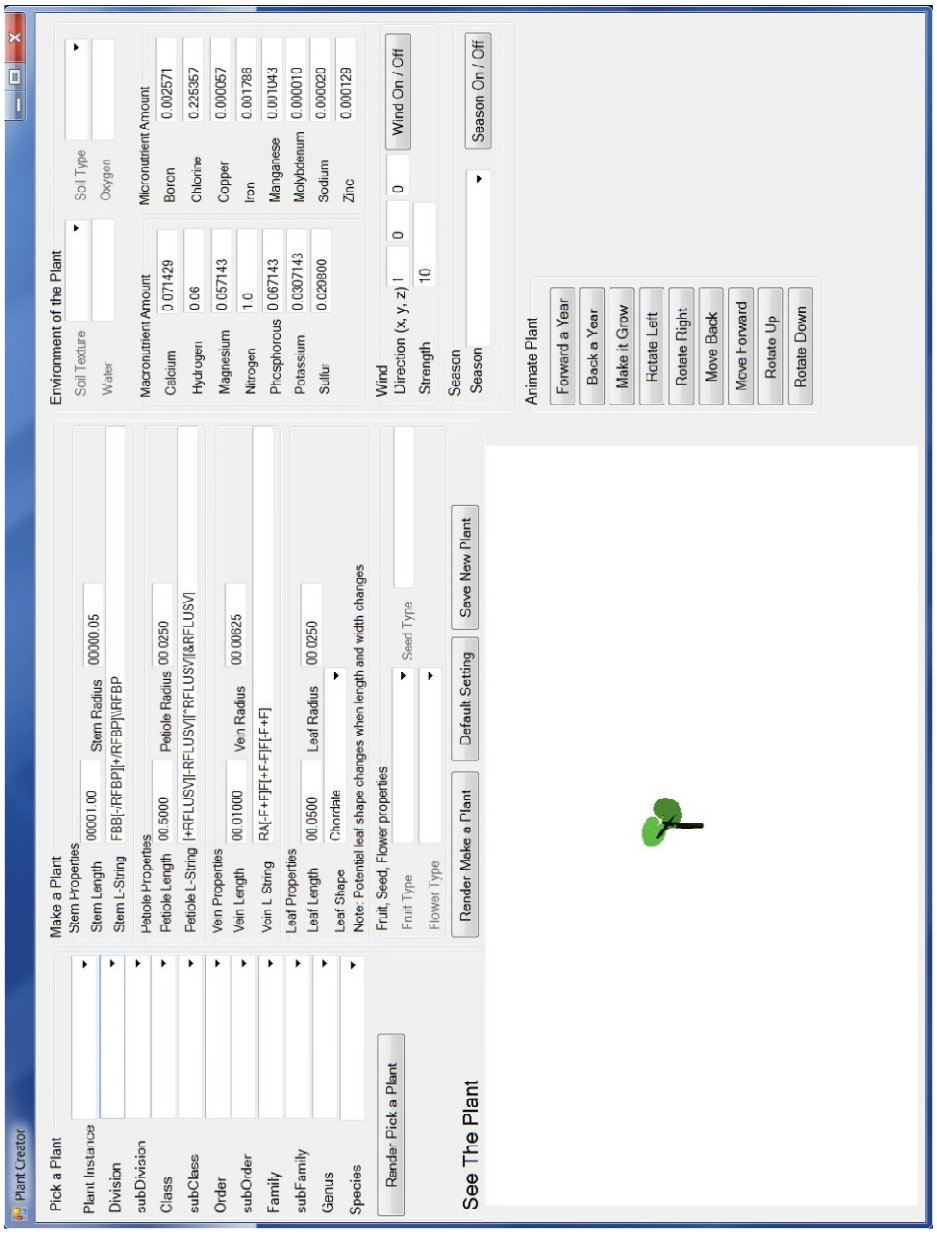


Figure 10: Software system implementation of PLANI

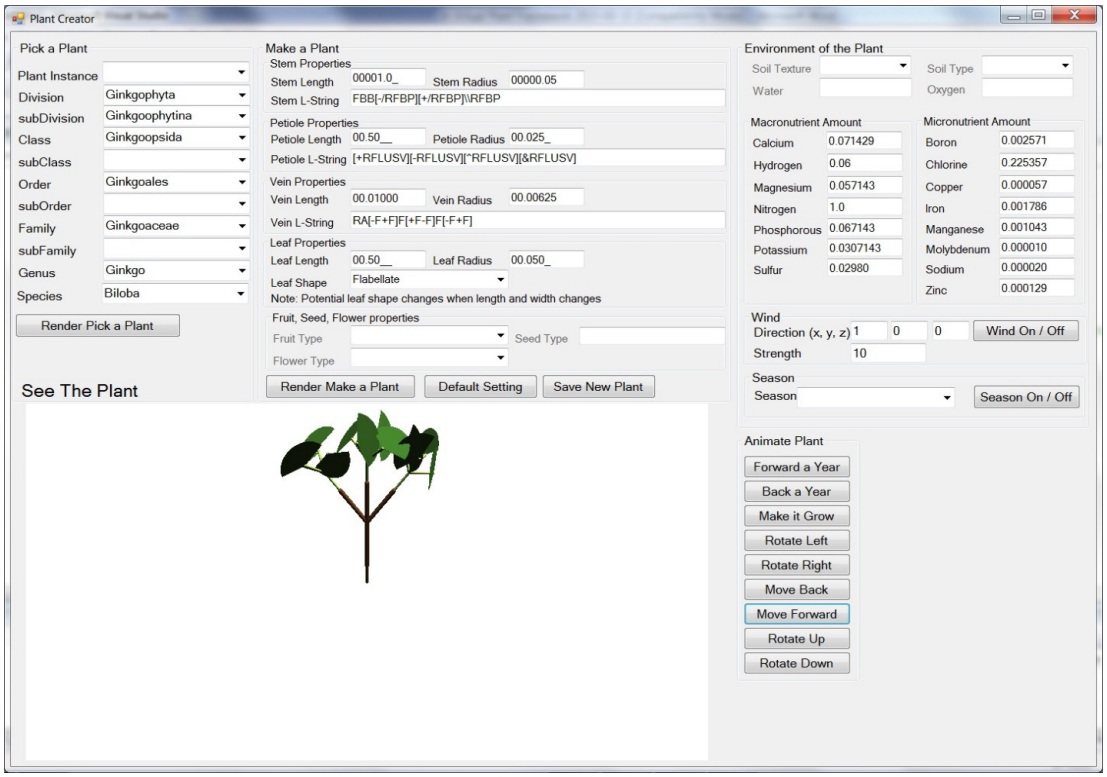


Figure 11: Species selection in Pick a Plant

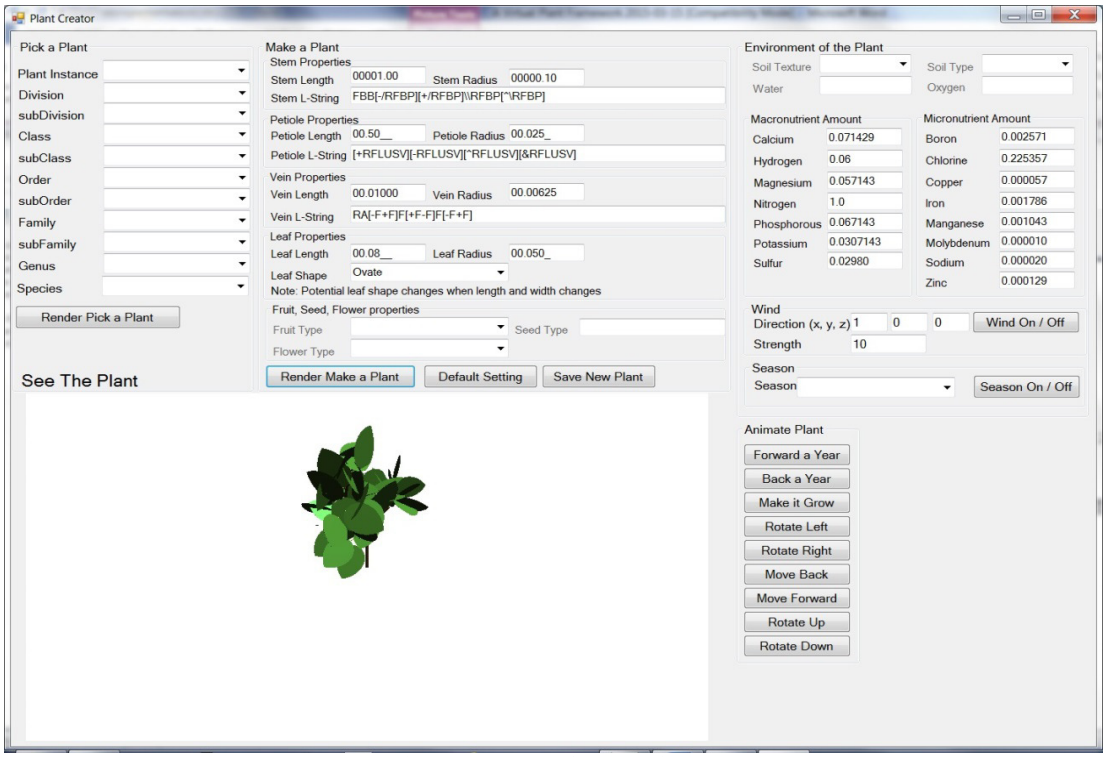


Figure 12: Tree with Oval leaves

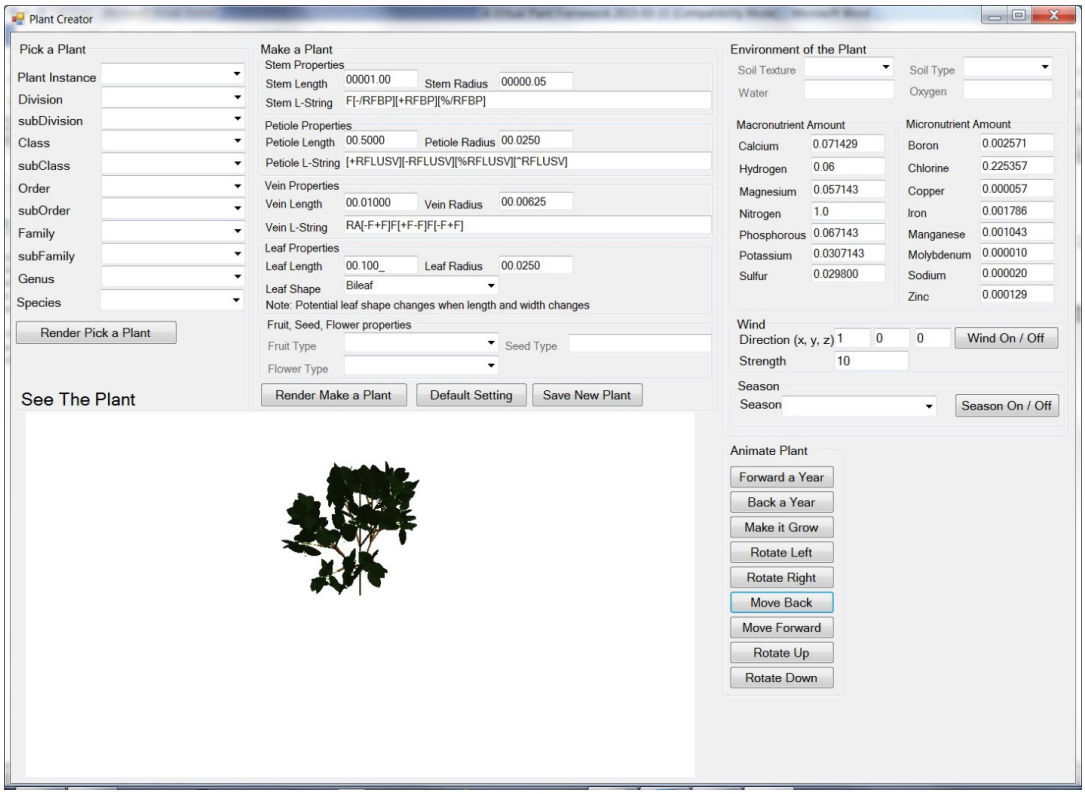


Figure 13: Tree with Compound Leaf

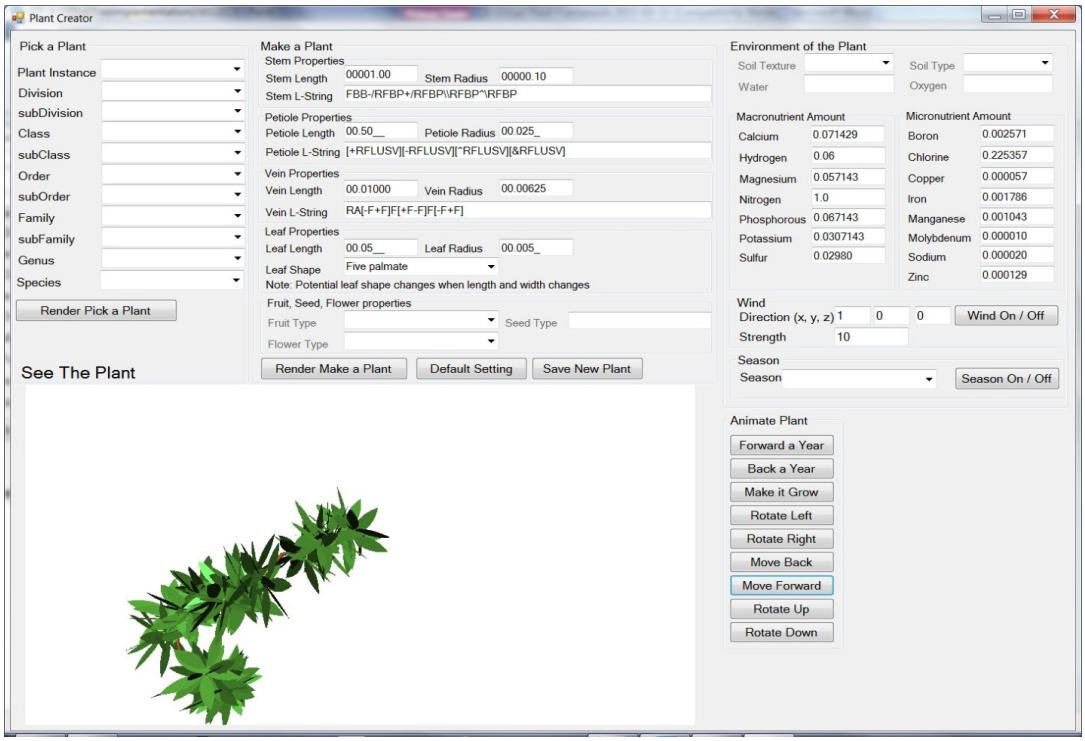


Figure 14: Vine example