CSIRO PUBLISHING
www.publish.csiro.au/journals/fpb *Functional Plant Biology*, 2008, **35**, 751–760

OpenAlea: a visual programming and component-based software platform for plant modelling

Christophe Pradal[A,D], *Samuel Dufour-Kowalski*[B], *Frédéric Boudon*[A], *Christian Fournier*[C] *and Christophe Godin*[B]

[A]CIRAD, UMR DAP and INRIA, Virtual Plants, TA A-96/02, 34398 Montpellier Cedex 5, France.
[B]INRIA, UMR DAP, Virtual Plants, TA A-96/02, 34398 Montpellier Cedex 5, France.
[C]INRA, UMR 759 LEPSE, 2 place Viala, 34060 Montpellier cedex 01, France.
[D]Corresponding author. Email: christophe.pradal@cirad.fr

This paper originates from a presentation at the 5th International Workshop on Functional–Structural Plant Models, Napier, New Zealand, November 2007.

Abstract. The development of functional–structural plant models requires an increasing amount of computer modelling. All these models are developed by different teams in various contexts and with different goals. Efficient and flexible computational frameworks are required to augment the interaction between these models, their reusability, and the possibility to compare them on identical datasets. In this paper, we present an open-source platform, OpenAlea, that provides a user-friendly environment for modellers, and advanced deployment methods. OpenAlea allows researchers to build models using a visual programming interface and provides a set of tools and models dedicated to plant modelling. Models and algorithms are embedded in OpenAlea 'components' with well defined input and output interfaces that can be easily interconnected to form more complex models and define more macroscopic components. The system architecture is based on the use of a general purpose, high-level, object-oriented script language, Python, widely used in other scientific areas. We present a brief rationale that underlies the architectural design of this system and we illustrate the use of the platform to assemble several heterogeneous model components and to rapidly prototype a complex modelling scenario.

Additional keywords: dataflow, interactive modelling, light interception, plant modeling, software architecture.

Introduction

Functional–structural plant models (FSPM) aim to simulate and help to understand the biological processes involved in the development and functioning of plants (Prusinkiewicz 2004; Godin and Sinoquet 2005; Vos *et al*. 2007). This requires efficiently using and combining models or computational methods from different scientific fields in order to analyse, simulate and understand complex plant processes at different scales (Prusinkiewicz and Hanan 2007). Owing to the different constraints and background of the teams, these models are developed using different programming languages, with different degrees of modularity and inter-operability. In addition, little attention is devoted to the reusability of the code and to its diffusion (packaging, installation procedures, website, portability to other operating systems, and documentation). This makes it difficult to exchange, re-use or combine models and simulation tools between teams (or even within a team). This may become particularly critical as the FSPM community wants to address the study of more and more complex systems, which requires integrating different models available from different groups at different scales.

Attempts have been made in the past to develop software platforms in the context of FSPM. The most popular is the L-Studio software, developed since the end of the 1980s by the group led by P. Prusinkiewicz (Prusinkiewicz and Lindenmayer 1990; Mech and Prusinkiewicz 1996). This platform runs on the Windows operating system and provides users with an integrated environment and a specific language called 'cpfg' dedicated to the modelling of plant development. This language was recently upgraded to L + C (based on the C++ programming language). This greatly extended the power of expression and the openness of the system.

A different user interface, 'VLab', has been designed by the same group to use 'cpfg' on Linux systems (Federl and Prusinkiewicz 1999). In itself, the VLab design is independent of the application domain. This interactive environment consists of experimental 'units' called objects, that encompass data files, and Linux programs, that operate on these data. To exchange data, objects must write the data to the disk. An inheritance mechanism allows objects to be refined using an object-oriented file system, and objects may be distributed in different locations across the web. Such features make it a powerful system for assembling pieces of code at a coarse grain level and for managing different versions of any given model. However, VLab uses of a shell language to combine stand-alone programs that have a low level

© CSIRO 2008 Open Access 10.1071/FP08084 1445-4408/08/100751

of interoperability, and does not allow easy control of data flows at a fine grain level due to the limited access that the modeller has to the internal data structures of the interconnected programs.

'GroIMP' (Kniemeyer *et al*. 2006) is another software platform based on L-systems, that was developed recently by W. Kurth and team in the context of plant modelling and simulation in biology. This open software platform is written in Java, which renders it independent of operating systems. Similarly to LStudio/VLab, GroIMP also relies on a special purpose language, 'XL', dedicated to the simulation of plants and, more generally, to the dynamic development of graph structures. The choice of Java as a programming language allows a tradeoff between an easy to use programming language (e.g. no pointers, automatic memory management) and a compiled efficient language such as C++.

Similarly to GroIMP but in a domain restricted to forest management, 'Capsis' is a computer platform based on Java (Goreaud *et al*. 2006), for studying forest practices that is worth mentioning in these approaches applied to plant modelling.

In a relatively different spirit, the 'AMAPmod' platform (Godin *et al*. 1997) focuses on plant architecture analysis rather than on plant growth simulation. It was originally based on a home-made language, 'AML', which was designed to provide a high degree of interaction between users and their models (Godin *et al*. 1999). The AML language was then abandoned and replaced by a more powerful language coming from the open software community, Python, which was found to achieve a very good compromise between interactivity, efficiency, stability, expressive power, and legibility both for expert programmers and beginners. This major upgrade of the AMAPmod system (now re-engineered as 'VPlants') initiated the development of OpenAlea.

Software platforms outside the world of plant modelling also inspired the development of OpenAlea. In particular, the use of visual programming was introduced in different projects: AVS in scientific visualisation (Upson *et al*. 1989), Vision (Sanner *et al*. 2002) in bioinformatics or Orange (Demsar *et al*. 2004) in data mining. This notion was shown to allow users natural access to the modelling system and easy sketching and reuse of model components.

We present, in this paper, the open-software platform, OpenAlea, for plant modelling based on a combination of the two families of approaches (i.e. plant architecture analysis and visual programming). OpenAlea is a flexible component-based framework designed to facilitate the integration and interoperability of heterogeneous models and data structures from different scientific disciplines at a fine grain level. Its architecture will also ease and accelerate the diffusion of new computational methods as they become available from different research groups. Such a software environment is targeted not only at developers and computer scientists but also at biologists, who may be able to assemble models while minimising the programming effort. The first section ('OpenAlea at a glance') presents a general outline of the OpenAlea platform. The second section details the design goals and requirements that drove the platform development. The third section describes the design choices and emphasises several critical technical issues. Finally, the last section provides an illustration of the use of the platform on a typical modelling application in the context of ecophysiology. This example shows how the platform can ease the integration and interoperation of heterogeneous software components in plant modelling applications.

OpenAlea at a glance

OpenAlea provides a graphical user interface (GUI), VisuAlea, which makes it possible to access easily the different components and functionalities of the system. It is composed of three main zones. The central zone (Fig. 1*B*) contains the graphical description of the model being built. The user can add or delete component nodes (in blue) and connect them via their input/output ports (yellow dots). Each component node contains parameters that can be edited through a specific GUI by clicking on the node. Component nodes available in the libraries installed on the user's computer can be browsed and selected using the package manager (Fig. 1*A*). Once the model is complete, the user can get the result of the model execution at any node by selecting this node and running it. The evaluation of a node changes its state which is represented by a colour. During the execution of the dataflow, the flow of node evaluation is, thus, represented by a flow of colour change. Depending on the type of the output data, the result is displayed by an appropriate graphical interface as a text, a graphic, or a 3-D scene (Fig. 1*D*). The result may also be exported to the Python interpreter for further use through the language (Fig. 1*C*). Figure 1 shows a small example in which a graphical model was designed to import the geometric models of a tulip and to multiply it using a component node representing a spatially uniform distribution.

Design goals and platform requirements

The OpenAlea platform was designed to meet the following requirements.

Ease of use

As stated above, OpenAlea proposes a visual programming environment and a collection of computational components, which make it simple to combine existing models in a new application. It also gives a simple multi-platform framework for the development and integration of components.

Reusability and extendibility

OpenAlea architecture aims at facilitating the solving of technical issues linked to sharing, reuse, and integration of software components, i.e. programs, algorithms and data structure from heterogeneous languages (mainly C, C++, Python, and Fortran). This makes the platform useful for multi-disciplinary projects and multi-scale modelling of plants.

Collaborative development

The development and ownership of OpenAlea are shared by various teams, and open to all the community. The overall software quality is improved by enforcing common rules and best practices. Synergy between multidisciplinary teams is also enhanced. The software life cycle is extended because the system is co-developed by different teams to suit their own needs. Economies of scale are achieved by sharing the costs of development, documentation and maintenance.

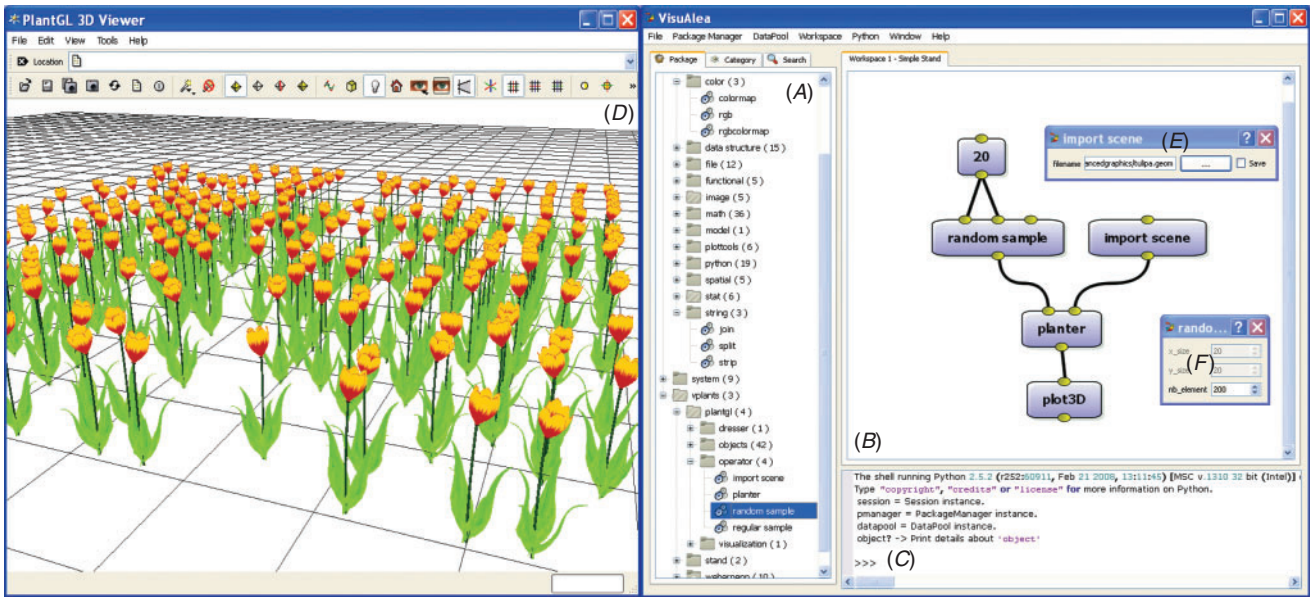

Fig. 1. Snapshot of the OpenAlea visual modelling environment. (*A*) The package manager list packages and nodes found on the system. (*B*) The graphical programming interface enables users to build visual dataflow by interconnecting nodes. A 3-D scene is built by associating a single geometry with a random distribution of points. (*C*) Low level interactions are done in the Python interpreter. (*D*) A 3-D viewer is directly called by the Plot3D component. (*E*, *F*) Widgets specific to each component are automatically generated.

Description of the platform

The OpenAlea architecture consists of: (a) a Python-language based system and a set of tools to integrate heterogeneous models implemented in various languages and on different platforms; (b) a component framework that allows dynamic management and composition of software components; (c) a visual-programming application for the interactive creation and control of complex models and for rapid prototyping; and (d) an environment for collaborative development and software diffusion.

Python-language based system and model integration

OpenAlea has been designed using a 'language-centric' approach (Sanner 1999) using the high-level, object-oriented Python script language as a framework. Script languages, like the Unix shell, have been successfully used for decades in the Unix world (Raymond 2003) to build flexible workflows from small stand-alone programs. Independent pieces of software can be combined via the language. New functionalities are easier to develop for users in an interpreted script language rather than in a compiled one. However, shell script languages require conversion of complex data structures into strings to support communication between programs. This may be inefficient for large data structures and requires extra work for developers to manage serialisation and marshalling methods. This limitation has been solved in other scientific packages [e.g. R (R Development Core Team 2007), Matlab (Higham and Higham 2005), and AMAPmod in plant modelling (Godin *et al*. 1997)] which have developed their own domain specific languages where common data structures are shared in memory. Among all scripts languages, the general purpose Python language was found to present unique key features. It is: (a) open source; (b) platform independent; (c) object-oriented; (d) user friendly; it has a simple-to-read syntax and is easy to learn, which allows even non-computer scientists to prototype rapidly new scripts or to transform existing ones (Ousterhout 1998; Ascher and Lutz 1999); (e) interactive: it allows direct testing of code without compilation process. The Python community is large and active, and a large number of scientific libraries are available (Oliphant 2007). Python framework enhances usability and inter-operability by providing a unique modelling language for heterogeneous software. It allows users to extend, compare, reuse and interconnect existing functionalities. It is used as a glue language between integrated components. Although the performance penalty is high for interpreted language compared with compiled language, performance bottlenecks in Python programs can be rewritten in compiled language for optimising speed. Existing C, C++ or Fortran programs and libraries can be imported as extension modules. For this, wrappers that specify how the components can be used in the Python language have to be implemented. Standard wrapping tools, such as Boost.Python (http://www.boost.org), Swig (http://www.swig.org, accessed 19 August 2008), and F2PY (http://www.scipy.org/F2PY, accessed 19 August 2008), are used to support this integration process. Transforming an existing library into a reusable component can also result in improvement in its design and programming interface. For this reason, we recommend the separation of different software functionality (e.g. data-structure, computational task, graphical representation) into different independent modules. This is intended to improve software quality and maintenance. However, the cost to obtain an overall quality improvement of software may be expensive in development time. A disadvantage of script language is that syntax errors are detected at run-time rather than at compile-time. To detect these errors early in the development process and to test the validity of the functionalities, unit-test suites can be

developed and source code checker can be used, like pylint (http://www.logilab.org, accessed 19 August 2008) and PyChecker (http://pychecker.sourceforge.net/, accessed 19 August 2008).

Component framework

OpenAlea implements the principles of a 'component framework' (Councill and Heineman 2001), which allows users to combine dynamically existing and independent pieces of software into customised workflows (Ludascher *et al*. 2006). This type of framework allows the decomposition of applications into separate and independent functional subsystems. Communication between components is achieved through interfaces (Szyperski 1998) and is explicitly represented graphically as connections between components.

The software relies on several key concepts: (a) a 'node' (Fig. 2) represents a software unit or 'logical component'. It is a function object which provides a certain type of service. It reads data on its input 'ports' and provides new data on its output ports. (b) A 'dataflow' (Johnston *et al*. 2004) is a graph composed of nodes connected by edges representing the flow of data from one node to the next. It defines a high level functional process well suited for coarse grain computation and close to natural algorithm design. (c) A 'composite-node' or 'macro node' is a node that encapsulates others nodes assembled in a dataflow and makes it possible to define a hierarchy of components. Node composition allows user to factorise common processes in a unique node and to create extended and reusable subsystems. (d) A 'package' is a deployment unit that contains a set of nodes, data as well as meta-information like authors, licence, institutes, version, category, description and documentation. (e) The 'package manager' allows for the dynamic search, loading and discovering of the functionalities by introspection of the available packages installed on the computer without requiring specific configuration. The platform modules and libraries are developed in a distributed way, and the availability of functionality depends on the user-defined system configuration.

Users can develop new functionalities that are added via the package manager at run-time without modification of the framework. The framework can be extended by combining nodes into composite-nodes or by implementing new functionality directly in Python at run-time using a code editor. Dataflows containing nodes and composite-nodes can be saved as standalone applications for end-users or as Python scripts.

In the dataflow, the nodes communicate by exchanging Python objects. An input and output port can be connected if their data types are compatible. Otherwise, an adaptor has to be inserted between the two nodes. A simple way to ensure input/output compatibility between heterogeneous components is to use the standard data type available in Python such as list or dictionary. For more complex types, such as graphs, some abstract interfaces are provided in OpenAlea to standardise and ease communication.

The evaluation of a dataflow is a recursive algorithm from a specific node selected by a user. All the nodes connected to its input ports are evaluated before evaluating the node itself. Cyclic dependencies in the graph are managed by setting the previously computed output values on the output ports or using default values for the first evaluation.

Visual programming

To enable scientists to build complex models without having to learn a textual programming language, we designed the visual programming environment, 'VisuAlea'. Using VisuAlea, the user can combine graphically different processing nodes provided by OpenAlea libraries and run the final scenario. The graphical models show clearly the dependencies between the processes as a graphical network and ease the understanding of

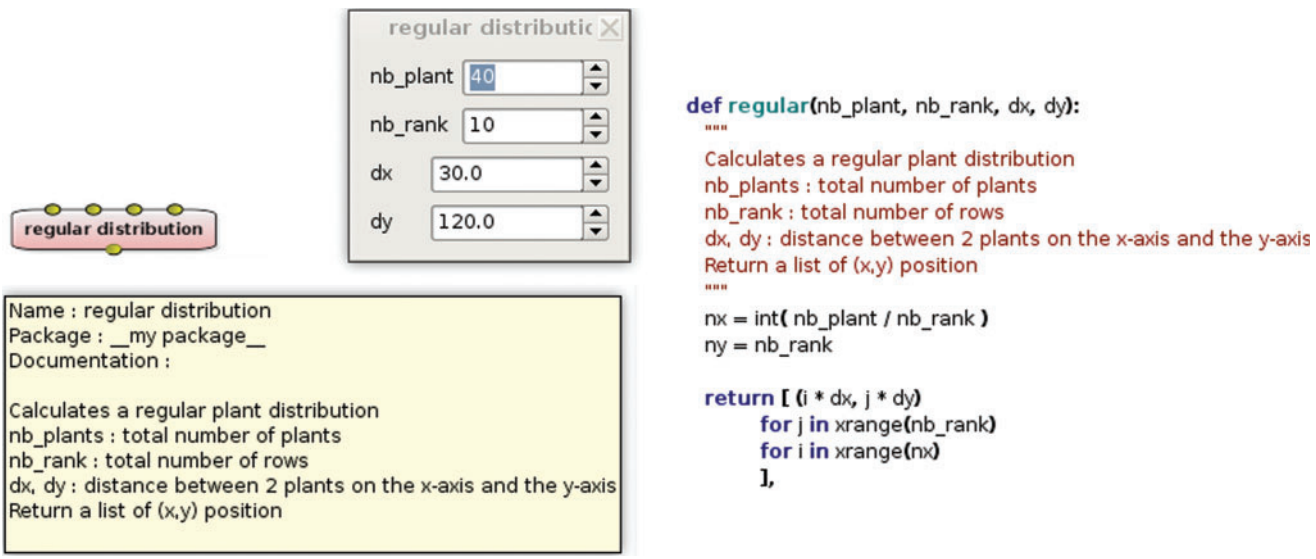

Fig. 2. A graphical node is a visual representation of a function. Input ports at the top represent the input arguments and output ports at the bottom, the resulting values. In this example, the 'regular' node generates a list of position (x, y) corresponding to a regular plant distribution. Documentation is automatically extracted and display in a tooltip. The node widget allows the user to set the value of the parameters. On the right, we show the related Python code.

the structure of the model. Users can interactively edit, save and compose nodes. In this visual approach, a graphical interface is associated with each node and enables the configuration and visualisation of their parameters and data. Customising parameters of the dataflow provides the user with an interactive way to explore and control the model. Complex components will have specifically designed dialogue boxes. For others, a dialogue box can be automatically generated according to the type of the input port. In this case, a widget catalogue provides common editors for simple types (e.g. integer, float, string, color, filename), 2-D and 3-D data plotters, sequence and graph editors. Thus, models that do not provide GUI can be easily integrated in the visual environment. Moreover, the catalogue can easily be extended with new widgets for new data types.

Advanced users may add new components by simply adding a Python function directly from VisuAlea. GUI and documentation are extracted and generated automatically. Finally, a Python shell has been integrated in the visual environment to give a flexible way for programmers to interact procedurally with the components and to extend their behaviour while taking advantage of the graphic representation of the data. VisuAlea favours the reuse of code and provides an environment for rapid prototyping.

In a standard modelling process, the modeller starts by creating a package in which (s)he can add components and a new dataflow. The dataflow can be saved in the package, or a subpart of the dataflow can be grouped into a composite node and saved to be re-used as a single node in a more complex dataflow or with different datasets.

To illustrate this principle, let us consider a set of nodes corresponding to a light interception model, inspired from the real case-study presented below:

- a node to read and construct a database of digitised points of a plant;
- a mesh reconstruction node, to calculate a triangle mesh representation of a plant from the digitised points;
- a light model node, to compute total light interception on a 3-D structures using data describing the light sources.

The dataflow in Fig. 3*A* shows a first connection of these nodes starting with a filename node for the digitised points and a parameter node for sky description. Eventually, this dataflow can be viewed as a more macroscopic model that implements a reusable functionality. In Fig. 3*B*, the different components are grouped to form the macro node 'composite light model' that can be tested with different parameters and reused in other dataflows. It is reused in the dataflow in Fig. 3*C* and tested on a set of sky parameters p_i, to explore, for instance, the response of the model to different lighting conditions. Resulting values are finally displayed on a 2-D plot.

Development environment and diffusion

For developers and modelling scientists, OpenAlea provides a set of software tools to build, package, install, and distribute their modules in a uniform way on multiple operating systems. It decreases development and maintenance costs whilst increasing software quality and providing a larger diffusion. In particular, some compilation and distribution tools make it possible with high level commands for users to avoid most of the problems due to platform specificity. Although pure Python components are natively platform independent, others have to be rebuilt and installed on each specific platform, which may be a rather complex task. To ease the compilation and deployment processes on multiple platforms, we have developed various tools such as SConsX and Deploy. SConsX is an extension package of SCons (Knight 2005). It simplifies the building of platform dependent packages by supporting different types of compilers (i.e. GCC, MinGW, Visual C++) and platform environments. Similarly, Deploy extends the standard Setuptools library for packaging and installation of modules by adding a support for reusable components with shared libraries. A graphical front-end of this tool has been developed to facilitate the install, update or removal of OpenAlea packages on Windows, Linux and Mac OS X platforms. The user selects the packages (s)he needs from a list of available packages. The selected packages and their dependencies are automatically downloaded and installed on the system. The list of available packages is retrieved from standard or user-defined web repositories (e.g. OpenAlea GForge public web repository or personal private repository using authentication). Third-party Python packages of the Python Package Index (PyPI, http://pypi.python.org, accessed 19 August 2008) are also accessible through this interface.

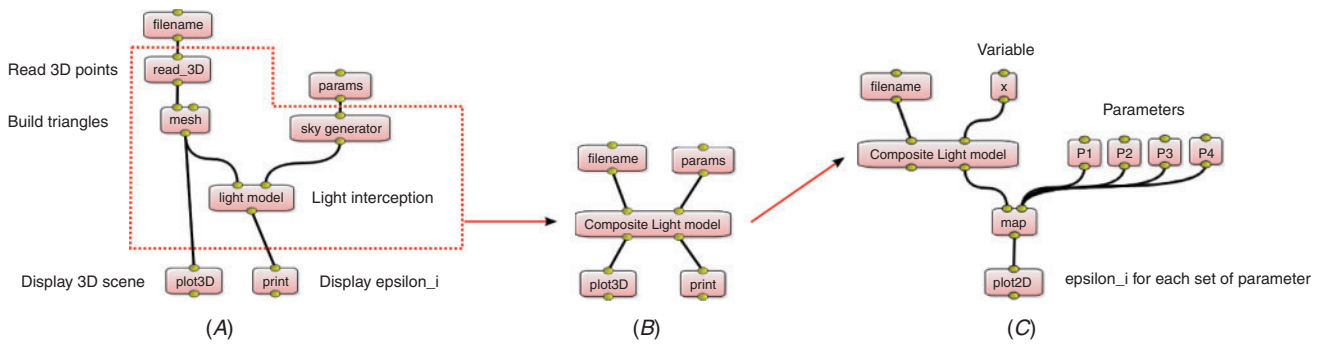

Fig. 3. (*A*) In the first example, we construct a plant model from a set of 3-D points read in a file. (*B*) Then, the light interception is computed using a sky description. The 3-D plant is displayed in a 3-D viewer, and the results of the light model are displayed in the shell. In the second example, the dataflow is simplified by grouping some nodes in a composite node. (*C*) The third example shows the same model applied for different set of parameters.

Some collaborative tools allow information, source codes, binaries and data to be shared and distributed over the internet. First, a collaborative website (http://openalea.gforge.inria.fr, accessed 19 August 2008) where the content is provided by users and developers makes it possible to share documentation and news. It offers access to the documentation (user tutorials, developer guides and general guidelines). A short presentation for each components distributed in OpenAlea is available and provided by the maintainer of the component. The website serves as a first medium of exchange between users, modellers and developers. Second, the project management and the distributed development of OpenAlea is made using a GForge server (http://gforge.inria.fr, accessed 19 August 2008) that contains amongst other things useful bug tracking and versioning tools for the source code.

The OpenAlea platform is distributed under an open source licence to foster collaborative development and diffusion. This licence allows external component developers to choose their own licence, including closed source ones. However, only open source components are distributed through the OpenAlea component repository. Selecting an open source licence for a component allows users to benefit for the support of the OpenAlea community such as: (i) compilation of binaries on different operating systems, (ii) easy access through the OpenAlea website and component repository, and (iii) possible improvement of the component by other teams which can provide bug fixes, documentation, and new features. The OpenAlea licence is also compatible with non-open-source ones and allows integration with proprietary modules. Users can also retrieve and share proprietary modules from private repositories in a secure and authenticated way using the deployment tools.

Currently integrated components

Several components have already been integrated to date in OpenAlea from different fields of plant modelling, such as plant architecture analysis, plant geometric modelling, ecophysiological processes, and meristem modelling and simulation (see Fig. 4).

(1) Plant architecture analysis: the VPlants package, successor of AMAPmod, provides data structure and algorithms to store, represent and explore multi-scale plant architectures. Statistical models like Hidden–Markov tree models (Durand *et al*. 2007) or change points detection models (Guédon *et al*. 2007) are provided to analyse branching pattern and tree architecture.
(2) Plant geometry modelling: the PlantGL graphic library (Pradal *et al*. 2007) contains a hierarchy of geometric objects dedicated to plant representations that can be assembled into a scene graph, a set of algorithms to manipulate them and some visualisation tools. Some parametric generative processes to build plant architecture (e.g. Weber and Penn 1995) are also integrated.
(3) Ecophysiological processes: Caribu (Chelle and Andrieu 1998) and RATP (Sinoquet *et al*. 2001) provide methods for light simulation in 3-D environments and for computing radiation interception, transpiration, and carbon gain of a tree canopy. The Drop model (Dufour-Kowalski *et al*. 2007) simulates rainfall interception and distribution by plants.
(4) Meristem modelling: mechanical models of tissue compute cell deformation and growth (Chopard *et al*. 2007).
(5) A catalogue component provides common tools for general purposes such as simple mathematical functions, standard data structures (e.g. string, list, dictionary), and file manipulation services.

A case-study of use of OpenAlea in ecophysiology: estimation by simulation of light interception efficiency

Overview

The objective of this case-study was to determine how the integral of the fraction of light intercepted by a maize (*Zea mays* L.) crop over the plant cycle is sensitive to natural variation in leaf shapes. To do so, the light interception efficiency (LIE) is estimated by a simulation procedure using different leaf shapes which were measured in the field for a given number of maize genotypes. This procedure required the use of three types of model: (i) a model of 3-D leaf shapes, (ii) a simulator of the development of the canopy, here ADEL-maize (Fournier and Andrieu 1998), and (iii) a radiative model, here Canestra (Chelle and Andrieu 1998).

Such a chain of models has already been developed and used several times (e.g. Fournier and Andrieu 1999; Pommel *et al*. 2001; Evers *et al*. 2007). However, the user had to re-use and adapt the existing models developed using different kinds of tools (R scripts for pre- and post-processing, Unix scripts and open-L-system scripts for simulation), which is not an easy task without the help of their authors. In this example, we show how OpenAlea helped setting up a more ergonomic, self-documented, re-usable and versatile application.

We detail hereafter how the three simulation tasks were embedded into independent functional components, and finally assembled using VisuAlea to get the final application (Fig. 5).

From field data to 3-D leaf shapes

Two properties of leaf shapes were measured: the variation of leaf width as a function of the distance from the base of the leaf, and the 3-D trajectory of the leaf midribs. In previous uses of ADEL-maize, an analytical model of leaf shape, i.e. composed of conic arcs (Prévot *et al*. 1991), was fitted to the data to smooth them out and remove digitising errors. The estimated parameters of this leaf model were used as inputs to the L-system based 3-D plant generator. In this case-study, we have developed a new parametric model because the shape of midrib leaf curves of certain genotypes presents several inflexion points which cannot be easily approximated using conics. This was not done before due to the difficulty to design new algorithm which used external scientific libraries. The midrib curve and the variation of the leaf width are approximated, in the parametric model, with NURBS curves using the least square fitting algorithm (Piegl and Tiller 1997), available in the Python scientific library, SciPy (Oliphant 2007). To optimise the final radiative computation, whose complexity depends on the square of the number of triangles of the leaves, the NURBS curves have been simplified as polylines with a given number of points using a decimation algorithm (Agarwal and Varadarajan 2000) developed in Python. Under VisuAlea (Fig. 5*A*), the user can graphically set the leaf data and control the level of discretisation of the final mesh

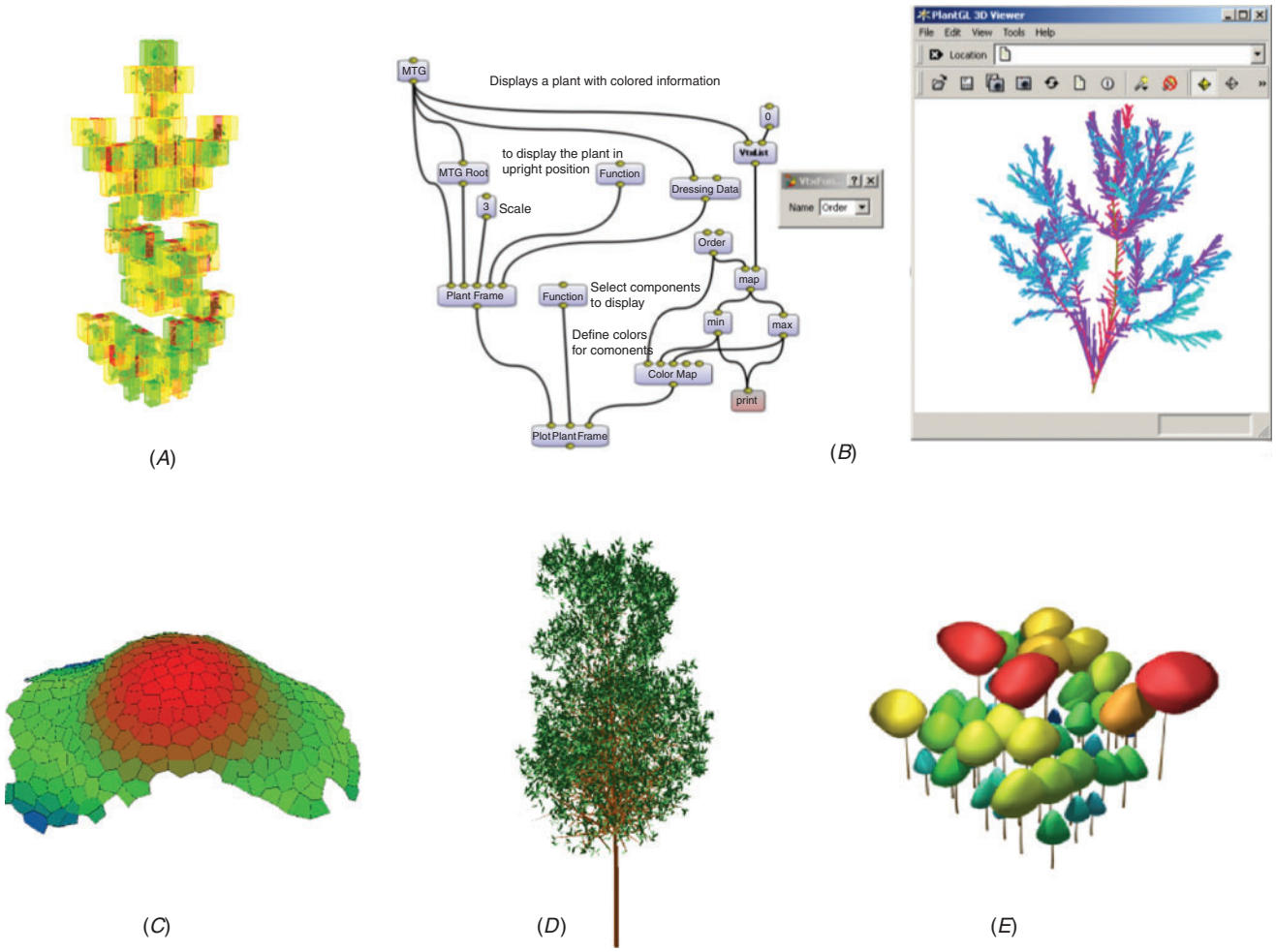

Fig. 4. Example of components integrated in OpenAlea. (*A*) Estimation of the fractal dimension of a plant foliage using the box counting method (Da Silva *et al*. 2006) (*B*) A visual programming example used to explore the topology and geometry of multiscale plant databases using VPlants components. (*C*) A 3-D surface tissue of a meristem. (*D*) Procedural generation of a tree architecture using the Weber and Penn algorithm. (*E*) A community of plants generated at the crown scale using the PlantGL component.

by setting the values of the 'fit leaves' nodes which convert the leaf measurement into simplified polylines. Using knowledge about maize leaf development (Fournier and Andrieu 1998), the leaf shape can be reconstructed at any stage of its development. To obtain the leaf shape from the curves and user-defined developmental parameters (e.g. length, radius), a PlantGL mesh is computed by sweeping a section line of length following the width variation along the approximated midrib curve. Such reconstruction was handled by the 'symbols' node (Fig. 5*A*; Point 4) and used during the geometric reconstruction of the plant.

From 3-D leaf shapes to canopy development

In previous applications, ADEL-maize, which is a cpfg script, was used to simulate directly canopy 3-D development. The simulation was done in two steps. First, the model computed the evolution of the topology and of the dimensions of the organs of each plant, and stored it as a string. Second, a 3-D mock-up of the canopy was computed using the cpfg interpretor and a homomorphism. In this application, we did not apply the homomorphism to be able to use the geometric leaf shapes built outside cpfg. The plant reconstruction was performed from the L-system string using LOGO style turtle interpretation (Prusinkiewicz 1986) implemented in PlantGL (Pradal *et al.* 2007). Finally, the resulting individual plant mock-ups were sent to a planter node that distributed the plants over a defined area.

From canopy reconstruction to LIE

LIE was computed with the radiative model Caribu, which is a package of OpenAlea. The model is itself composed of several programs that can be arranged to fit particular needs. We used one of the arrangements that computes first order interception for an overcast sky, issued in the package in the form of a VisuAlea dataflow. We simply saved this Caribu dataflow as a composite node, imported it to the Adel dataflow (Fig. 5*A*), and made connections between slots. This package also already included visualisation tools based on PlantGL (such as the one producing

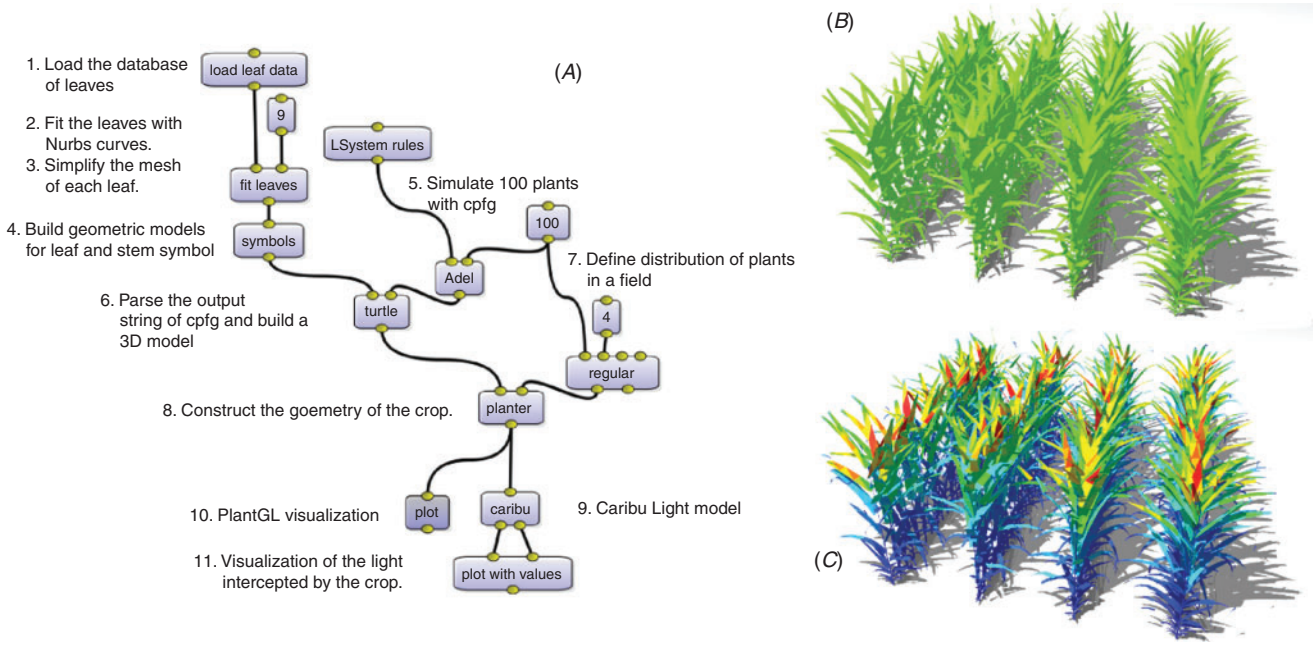

Fig. 5. Snapshots of (*A*) the VisuAlea dataflow, and (*B*) two outputs of an application allowing to reconstruct a maize canopy, and (*C*) to estimate light distribution within it. Annotations on the dataflow succinctly describe the functions of the different nodes. Nodes 1–4 defines the leaf shape model, which is a function that returns leaf shape at a given stage of development, from a set of curves fitted to digitise mature leaf shape data. Node 5 is an L-system engine simulating plant development from an L-system script ('LSystemRules'). Nodes 6–8 are for the reconstruction of the 3-D scene: one node combines the L-system output with the leaf model to reconstruct the plants ('turtle'), and one node ('planter') is used for placing plants according to a pattern ('regular'). Node 9 is for the radiative model, and nodes 10 and 11 are for producing 3-D outputs (*B*, *C*). Three parameters are represented with nodes to allow a direct interaction with the application: the number of polygons used to represent leaves (9), the total number of plants in the scene (100) and the number of rows (4).

output in Fig. 5*C*) and post-treatment routines for computing LIE. The complete dataflow (Fig. 5*A*) could be saved as a composite node and used in a new dataflow that iterates on different input datasets (similar to Fig. 4).

In this application, OpenAlea was used to extend the capabilities of the original application and to re-implement it in a more modular way, while improving the clarity of the chaining of the models. The ADEL application has inherited new features from the use of already existing tools. These new features include: (i) a parametric model to represent leaf shapes using parametric surfaces computed directly from digitised leaves; (ii) user control of the number of polygons used to represent leaf shapes, and (iii) access to a large palette of sowing strategies. Visualisation and plotting tools are provided by PlantGL to generate different kinds of outputs (e.g. images, animations). Although the dataflow presented in Fig. 5*A* is specific to this particular application, it is easily editable and configurable for other objectives. For example, we can easily imagine replacing the maize model by another plant model, even developed with another simulator. All this finally requires a very limited programming effort, because of the re-use of libraries, and the automatic generation of graphical interfaces under VisuAlea.

Conclusion

The major achievement of OpenAlea is to provide a visual and interactive interface to the inner structure of an FSPM application. This greatly improves the potential of sharing and reusing specialised integrated models, since embedded submodels, data-structures, or algorithms can be recomposed or combined to fit different modelling objectives. This also increases the end users' knowledge of how an application works, by allowing independent evaluation of any part of the model dataflow. As OpenAlea is primarily intended for the FSPM community, we hope that such a platform will facilitate the emergence and sharing of generic components and algorithms able to perform standard modelling tasks in this domain. We also paid a particular attention to providing tools to ease the integration of existing models, so that a large community of scientists could use and 'feed' the platform. In its present state, OpenAlea is suited to build examples like the one presented here, where individual components have to be chained sequentially, and with a genericity of algorithms at the level of model subunit. The visual programming environment has been designed for model integration and connection rather than for modelling feedback and retroaction between models. It has been based on a dataflow model of computation where control flow and feedback are difficult to represent, like in functional languages. However, retro-action and feedback can be managed within specific nodes such as simulation nodes or biophysical solvers. OpenAlea only partially addresses the question, pointed out by Prusinkiewicz *et al.* (2007), regarding the construction of comprehensive models that incorporate several aspects of plant functioning with intricate interactions between functions (for example, a plant development model coupled with hormonal control, partitioning of resources, water fluxes and biomechanics). This would probably require one to define and share generic data structures representing the plant on different

scales, and address, both theoretically and algorithmically, the problem of simulating different processes acting in parallel at different scales.

A first step, might be, more modestly, to start connections between OpenAlea and other major software platforms dedicated to FSPM simulations (e.g. LStudio/Vlab, GroIMP) in order to identify current limitations and start defining data standards and databases that can be shared by the plant modelling community.

Acknowledgements

The authors thank Mrs and Mr Hopkins for editorial help, and the two anonymous reviewers for their constructive criticism. This research has been supported by the developer community of OpenAlea, by grants from INRIA, CIRAD, and INRA (Réseau Ecophysiologique de l'Arbre), and by the ANR project NatSim.

References

Agarwal PK, Varadarajan KR (2000) Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry* **23**, 273–291. doi: 10.1007/PL00009500

Ascher D, Lutz M (1999) 'Learning Python.' (O'Reilly and Associates: Sebastopol, CA)

Chelle M, Andrieu B (1998) The nested radiosity model for the distribution of light within plant canopies. *Ecological Modelling* **111**, 75–91. doi: 10.1016/S0304-3800(98)00100-8

Chopard J, Godin C, Traas J (2007) Toward a formal expression of morphogenesis: a mechanics based integration of cell growth at tissue scale. In 'Proceedings of the 7th International Workshop on Information Processing in Cells and Tissues. IPCAT'. pp. 388–399. (Crook N and Scheper T: Oxford, UK)

Councill B, Heineman G T (2001) Definition of a software component and its elements. In 'Component-based software engineering: putting the pieces together'. pp. 5–19. (Addison–Wesley Longman Publishing Co. Inc.: Boston, MA)

Da Silva D, Boudon F, Godin C, Puech O, Smith C, Sinoquet H (2006) A critical appraisal of the box counting method to assess the fractal dimension of tree crowns. *Lecture Notes in Computer Science* **4291**, 751–760. doi: 10.1007/11919476_75

Demsar J, Zupan B, Leban G (2004) Orange: from experimental machine learning to interactive data mining. White Paper, Faculty of Computer and Information Science, University of Ljubljana.

Dufour-Kowalski S, Bassette C, Bussière F (2007) A software for the simulation of rainfall distribution on 3-D plant architecture: PyDrop. In 'Proceedings of the 5th International Workshop on Functional–structural Plant Models'. (Eds P Prusinkiewicz, J Hanan, B Lane) pp. 29.1–29.3. (HortResearch: Auckland, New Zealand)

Durand JB, Caraglio Y, Heuret P, Nicolini E (2007) Segmentation-based approaches for characterising plant architecture and assessing its plasticity at different scales. In 'Proceedings of the 5th international workshop on functional–structural plant models'. (Eds P Prusinkiewicz, J Hanan, B Lane) pp. 39.1–39.3. (HortResearch: Auckland, New Zealand)

Evers JB, Vos J, Chelle M, Andrieu B, Fournier C, Struik PC (2007) Simulating the effects of localized red: far-red ratio on tillering in spring wheat (*Triticum aestivum*) using a three-dimensional virtual plant model. *New Phytologist* **176**, 325–336. doi: 10.1111/j.1469-8137.2007.02168.x

Federl P, Prusinkiewicz P (1999) Virtual laboratory: an interactive software environment for computer graphics. In 'Proceedings of Computer Graphics International'. pp. 93–100. (IEEE Computer Society: Washington, DC)

Fournier C, Andrieu B (1998) A 3-D architectural and process-based model of maize development. *Annals of Botany* **81**, 233–250. doi: 10.1006/anbo.1997.0549

Fournier C, Andrieu B (1999) ADEL-maize: an L-system based model for the integration of growth processes from the organ to the canopy. Application to regulation of morphogenesis by light availability. *Agronomie* **19**, 313–327. doi: 10.1051/agro:19990311

Godin C, Sinoquet H (2005) Functional–structural plant modelling. *New Phytologist* **166**, 705–708. doi: 10.1111/j.1469-8137.2005.01445.x

Godin C, Costes E, Caraglio Y (1997) Exploring plant topological structure with the AMAPmod software: an outline. *Silva Fennica* **31**, 355–366.

Godin C, Costes E, Sinoquet H (1999) A method for describing plant architecture which integrates topology and geometry. *Annals of Botany* **84**, 343–357. doi: 10.1006/anbo.1999.0923

Goreaud F, Alvarez I, Courbaud B, de Coligny F (2006) Long-term influence of the spatial structure of an initial state on the dynamics of a forest growth model: a simulation study using the Capsis platform. *Simulation* **82**, 475–495. doi: 10.1177/0037549706070397

Guédon Y, Caraglio Y, Heuret P, Lebarbier E, Meredieu C (2007) Identifying and characterizing the ontogenetic component in tree development. In 'Proceeding of the 5th International Workshop on Functional–structural Plant Models'. (Eds P Prusinkiewicz, J Hanan, B Lane) pp. 38.1–38.5. (HortResearch: Auckland, New Zealand)

Higham DJ, Higham NJ (2005) 'MATLAB Guide SIAM: Society for Industrial and Applied Mathematic.' (Society for Industrial and Applied Mathematics: Philadelphia, PA)

Johnston WM, Hanna JRP, Millar RJ (2004) Advances in dataflow programming languages. *ACM Computing Surveys* **36**, 1–34. doi: 10.1145/1013208.1013209

Kniemeyer O, Buck-Sorlin G, Kurth W (2006) GroIMP as a platform for functional–structural modelling of plants. In 'Functional–structural plant modelling in crop production'. pp. 43–52. (Springer-Verlag: Dordrecht, The Netherlands)

Knight S (2005) Building software with Scons. *Computing in Science & Engineering* **7**, 79–88. doi: 10.1109/MCSE.2005.11

Ludascher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J, Zhao Y (2006) Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* **18**, 1039–1065. doi: 10.1002/cpe.994

Mech R, Prusinkiewicz P (1996) Visual models of plants interacting with their environments. In 'SIGGRAPH '96'. (Ed. H Rushmeier) pp. 397–410. (Addison-Wesley: New York)

Oliphant TE (2007) Python for scientific computing. *Computing in Science & Engineering* **9**, 10–20. doi: 10.1109/MCSE.2007.58

Ousterhout JK (1998) Scripting: higher-level programming for the 21st century. *Computer* **31**, 23–30. doi: 10.1109/2.660187

Piegl L, Tiller W (1997) 'The Nurbs book.' (Springer-Verlag: New York)

Pommel B, Sohbi Y, Andrieu B (2001) Use of virtual 3-D maize canopies to assess the effect of plot heterogeneity on radiation interception. *Agricultural and Forest Meteorology* **110**, 55–67. doi: 10.1016/S0168-1923(01)00270-2

Pradal C, Boudon F, Nouguier C, Chopard J, Godin C (2007) PlantGL: a Python-based geometric library for 3-D plant modelling at different scales. INRIA Research Report. (INRIA: Sophia Antipolis, France)

Prévot L, Aries F, Monestiez P (1991) Modélisation de la structure géométrique du maïs. *Agronomie* **11**, 491–503. doi: 10.1051/agro:19910606

Prusinkiewicz P (2004) Art and science for life: designing and growing virtual plants with L-systems. *Acta Horticulturae* **630**, 15–28.

Prusinkiewicz P, Hanan J (2007) 'Proceedings of the 4th International Workshop on Functional–structural Plant Models, FSPM05.' (HortResearch: Napier, New Zealand)

Prusinkiewicz P, Lindenmayer A (1990) 'The algorithmic beauty of plants.' (Springer-Verlag: New York)

Prusinkiewicz P, Karkowski R, Lane B (2007) The L + C plant-modelling language. In 'Functional–structural plant modelling in crop production'. pp. 27–42. (Springer-Verlag: Dordrecht, The Netherlands)

R Development Core Team (2007) 'An introduction to R.' (Network Theory Limited: Bristol, UK)

Raymond ES (2003) 'The art of Unix programming.' (Pearson Education: UK)

Sanner MF (1999) Python: a programming language for software integration and development. *Journal of Molecular Graphics & Modelling* **17**, 57–61.

Sanner MF, Stoffler D, Olson AJ (2002) ViPEr, a visual programming environment for Python. In 'Proceedings of the 10th International Python Conference'. pp. 103–115.

Sinoquet H, Roux XL, Adam B, Ameglio T, Daudet FA (2001) RATP: a model for simulating the spatial distribution of radiation absorption, transpiration and photosynthesis within canopies: application to an isolated tree crown. *Plant, Cell & Environment* **24**, 395–406. doi: 10.1046/j.1365-3040.2001.00694.x

Szyperski C (1998) 'Component software: beyond object-oriented programming.' (Addison-Wesley: Harlow, England)

Upson C, Faulhaber TA, Kamins D, Laidlaw D, Schlegel D, Vroom J, Gurwitz R, van Dam A (1989) The application visualization system: a computational environment for scientific visualization. *Computer Graphics and Applications* **9**, 30–42. doi: 10.1109/38.31462

Vos J, Marcelis LFM, De Visser PHB, Struik PC, Evers JB (2007) 'Functional–structural plant modelling in crop production.' (Springer-Verlag: Dordrecht, The Netherlands)

Weber J, Penn J (1995) Creation and rendering of realistic trees. In 'Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques'. pp. 119–128. (ACM Press: New York)

Manuscript received 18 March 2008, accepted 29 July 2008