



HAL
open science

OpenAlea: A visual programming and component-based software platform for plant modeling

Christophe Pradal, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier, Christophe Godin

► To cite this version:

Christophe Pradal, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier, Christophe Godin. OpenAlea: A visual programming and component-based software platform for plant modeling. *Functional Plant Biology*, 2008, 35 (10), pp.751-760. 10.1071/FP08084 . hal-00831811

HAL Id: hal-00831811

<https://inria.hal.science/hal-00831811>

Submitted on 7 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

1 **OpenAlea: A visual programming and component-based**
2 **software platform for plant modeling**

3
4 Christophe Pradal^A, Samuel Dufour-Kowalski^B, Frédéric Boudon^A, Christian Fournier^C,

5 Christophe Godin^B

6 ^A CIRAD, UMR DAP and INRIA, Virtual Plants

7 TA A-96/02, 34398 Montpellier Cedex 5, France

8 ^B INRIA, UMR DAP, Virtual Plants

9 TA A-96/02, 34398 Montpellier Cedex 5, France

10 ^C INRA, UMR 759 LEPSE, 2 place Viala, 34060 Montpellier cedex 01, France

11
12 **Keywords:** plant modeling, software architecture, interactive modeling, dataflow, light
13 interception

14
15 **Abstract**

16 As illustrated by the approaches presented during the 5th FSPM workshop (Prusinkiewicz
17 and Hanan 2007, and this issue), the development of functional-structural plant models
18 requires an increasing amount of computer modeling. All these models are developed by
19 different teams in various contexts and with different goals. Efficient and flexible
20 computational frameworks are required to augment the interaction between these models,
21 their reusability, and the possibility to compare them on identical datasets.

22 In this paper, we present an open-source platform, OpenAlea, that provides a user-friendly
23 environment for modelers, and advanced deployment methods. OpenAlea allows researchers

24 to build models using a visual programming interface and provides a set of tools and models
25 dedicated to plant modeling. Models and algorithms are embedded in OpenAlea *components*
26 with well defined input and output interfaces that can be easily interconnected to form more
27 complex models and define more macroscopic components. The system architecture is
28 based on the use of a general purpose, high-level, object-oriented script language, Python,
29 widely used in other scientific areas. We briefly present the rationale that underlies the
30 architectural design of this system and we illustrate the use of the platform to assemble
31 several heterogeneous model components and to rapidly prototype a complex modeling
32 scenario.

33

34 **Introduction**

35 Functional-structural plant models (FSPM) aim to simulate and help to understand the
36 biological processes involved in the development and functioning of plants (Prusinkiewicz
37 2004; Godin and Sinoquet 2005; Vos *et al.* 2007). This requires efficiently using and
38 combining models or computational methods from different scientific fields in order to
39 analyze, simulate and understand complex plant processes at different scales. Due to the
40 different constraints and background of the teams, these models are developed using
41 different programming languages, with different degrees of modularity and inter-
42 operability. In addition, little attention is devoted to the reusability of the code and to its
43 diffusion (packaging, installation procedures, web site, portability to other operating
44 systems, and documentation). This makes it difficult to exchange, re-use or combine
45 models and simulation tools between teams (or even within a team). This may become
46 particularly critical as the FSPM community wants to address the study of more and more

47 complex systems, which requires integrating different models available from different
48 groups at different scales.

49 Attempts have been made in the past to develop software platforms in the context of
50 FSPM. The most popular is the L-Studio software, developed since the end of the 80's by
51 the group led by P. Prusinkiewicz (Prusinkiewicz and Lindenmayer 1990; Mech and
52 Prusinkiewicz 1996). This platform runs on the Windows operating system and provides
53 users with an integrated environment and a specific language called *cpfg* dedicated to the
54 modeling of plant development. This language was recently upgraded to *L+C* (based on the
55 C++ programming language). This greatly extended the power of expression and the
56 openness of the system.

57 A different user interface, *VLab*, has been designed by the same group to use *cpfg* on Linux
58 systems (Federl and Prusinkiewicz 1999). In itself, the *VLab* design is independent of the
59 application domain. This interactive environment consists of experimental *units* called
60 objects, that encompass data files, and Linux programs, that operate on these data. To
61 exchange data, objects must write the data to the disk. An inheritance mechanism allows
62 objects to be refined using an object-oriented file system, and objects may be distributed in
63 different locations across the web. Such features make it a powerful system for assembling
64 pieces of code at a coarse grain level and for managing different versions of any given
65 model. On the other hand, *VLab* uses of a shell language to combine stand-alone programs
66 that have a low level of interoperability, does not allow easy control of data flows at a fine
67 grain level due to the limited access that the modeler has to the internal data structures of
68 the interconnected programs.

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

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

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

89 Software platforms outside the world of plant modeling also inspired the development of
90 OpenAlea. In particular, the use of visual programming was introduced in different
91 projects: AVS in scientific visualization (Upson *et al.* 1999), Vision (Sanner 2002) in

92 bioinformatics or Orange (Demsar *et al.*, 2004) in data-mining. This notion was shown to
93 allow users natural access to the modeling system and easy sketching and reuse of model
94 components.

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

110

111 **OpenAlea at a glance**

112

113 OpenAlea provides a graphical user interface (GUI), VisuAlea, which makes it possible to
114 access easily the different components and functionalities of the system. It is composed of

115 three main zones. The central zone (Figure 1.B) contains the graphical description of the
116 model being built. The user can add or delete component nodes (in blue) and connect them
117 via their input/output ports (yellow dots). Each component node contains parameters that
118 can be edited through a specific GUI by clicking on the node. Component nodes available
119 in the libraries installed on the user's computer can be browsed and selected using the
120 package manager (Figure 1.A). Once the model is complete, the user can get the result of
121 the model execution at any node by selecting this node and running it. The evaluation of a
122 node changes its state which is represented by a color. During the execution of the
123 dataflow, the flow of node evaluation is thus represented by a flow of color change.
124 Depending on the type of the output data, the result is displayed by an appropriate
125 graphical interface as a text, a graphic, or a 3D scene (Figure 1.D). The result may also be
126 exported to the Python interpreter for further use through the language (Figure 1.C). Figure
127 1 shows a small example in which a graphical model was designed to import the geometric
128 models of a tulip and to multiply it using a component node representing a spatially
129 uniform distribution.

130

131 **Design goals and platform requirements**

132 The OpenAlea platform was designed to meet the following requirements:

133

134 **Ease of use.** As stated above, OpenAlea proposes a visual programming environment and a
135 collection of computational components, which make it simple to combine existing models

136 in a new application. It also gives a simple multi-platform framework for the development
137 and integration of components.

138

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

144

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

151

152 **Description of the platform**

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

159 *Python-language based system and Model integration*

160 OpenAlea has been designed using a “language-centric” approach (Sanner 1999) using the
161 high-level, object-oriented Python script language as a framework. Script languages, like
162 the Unix shell, have been successfully used for decades in the Unix world (Raymond 2004)
163 to build flexible workflows from small stand-alone programs. Independent pieces of
164 software can be combined via the language. New functionalities are easier to develop for
165 users in an interpreted script language rather than in a compiled one. However, shell script
166 languages require conversion of complex data structures into strings to support
167 communication between programs. This may be inefficient for large data structures and
168 requires extra-work for developers to manage serialization and marshalling methods. This
169 limitation has been solved in other scientific packages (e.g. R (R Development Core Team
170 2007), Matlab (Higham and Higham 2005), and AMAPmod in plant modeling (Godin *et al.*
171 1997)) which have developed their own domain specific languages where common data
172 structures are shared in memory. Among all scripts languages, the general purpose Python
173 language was found to present unique key features. It is (a) open-source; (b) platform
174 independent; (c) object-oriented; (d) user-friendly; it has a simple-to-read syntax and is

175 easy to learn, which allows even non computer scientists to prototype rapidly new scripts or
176 to transform existing ones (Asher and Lutz 1999, Ousterhout 1998); (e) interactive: it
177 allows direct testing of code without compilation process. The Python community is large
178 and active, and a large number of scientific libraries are available (Oliphant 2007). Python
179 framework enhances usability and inter-operability by providing a unique modeling
180 language for heterogeneous software. It allows to extend, compare, reuse and interconnect
181 existing functionalities. It is used as a glue language between integrated components.
182 Although the performance penalty is high for interpreted language compared to compiled
183 language, performance bottlenecks in Python programs can be rewritten in compiled
184 language for optimizing speed. Existing C, C++ or Fortran programs and libraries can be
185 imported as extension modules. For this, wrappers that specify how the components can be
186 used in the Python language have to be implemented. Standard wrapping tools, such as
187 Boost.Python (<http://www.boost.org>), Swig (<http://www.swig.org>), and F2PY
188 (<http://www.scipy.org/F2PY>), are used to support this integration process. Transforming an
189 existing library into a reusable component can also result in improvement in its design and
190 programming interface. For this reason, we recommend the separation of different software
191 functionality (e.g. data-structure, computational task, graphical representation, *etc.*) into
192 different independent modules. This is intended to improve software quality and
193 maintenance. However, the cost to obtain an overall quality improvement of software may
194 be expensive in development time. A disadvantage of script language is that syntax errors
195 are detected at run-time rather than at compile-time. To detect these errors early in the
196 development process and to test the validity of the functionalities, unit-test suites can be
197 developed and source code checker can be used, like pylint (<http://www.logilab.org>) and
198 PyChecker (<http://pychecker.sourceforge.net/>).

199

200 *Component framework*

201 OpenAlea implements the principles of a *component framework* (Council and Heineman
202 2001), which allows users to combine dynamically existing and independent pieces of
203 software into customized workflows (Ludascher *et al.* 2006). This type of framework
204 allows the decomposition of applications into separate and independent functional
205 subsystems. Communication between components is achieved through interfaces
206 (Szyperski 1998) and is explicitly represented graphically as connections between
207 components.

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

222 distributed way, and the availability of functionality depends on the user-defined system
223 configuration.

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

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

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

239

240 *Visual Programming*

241 To enable scientists to build complex models without having to learn a textual
242 programming language, we designed the visual programming environment, *VisuAlea*.
243 Using *VisuAlea*, the user can combine graphically different processing nodes provided by
244 OpenAlea libraries and run the final scenario. The graphical models show clearly the

245 dependencies between the processes as a graphical network and ease the understanding of
246 the structure of the model. Users can interactively edit, save and compose nodes. In this
247 visual approach, a graphical interface is associated with each node and enables the
248 configuration and visualization of their parameters and data. Customizing parameters of the
249 dataflow provides the user with an interactive way to explore and control the model.
250 Complex components will have specifically designed dialog boxes. For others, a dialog box
251 can be automatically generated according to the type of the input port. In this case, a widget
252 catalog provides common editors for simple types (e.g. integer, float, string, color,
253 filename, etc.), 2D and 3D data plotters, sequence and graph editors. Thus, models that do
254 not provide GUI can be easily integrated in the visual environment. Moreover, the catalog
255 can easily be extended with new widgets for new data types.

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

262

263 In a standard modeling process, the modeler starts by creating a package in which (s)he can
264 add components and a new dataflow. The dataflow can be saved in the package, or a sub-
265 part of the dataflow can be grouped into a composite node and saved to be re-used as a
266 single node in a more complex dataflow or with different data sets.

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

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

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

282

283 *Development environment and diffusion*

284 For developers and modeling scientists, OpenAlea provides a set of software tools to build,
285 package, install, and distribute their modules in a uniform way on multiple operating
286 systems. It decreases development and maintenance costs whilst increasing software
287 quality and providing a larger diffusion. In particular, some compilation and distribution
288 tools make it possible with high level commands for users to avoid most of the problems
289 due to platform specificity. While pure Python components are natively platform
290 independent, others have to be rebuilt and installed on each specific platform, which may
291 be a rather complex task. To ease the compilation and deployment processes on multiple

292 platforms, we have developed various tools such as SConsX and Deploy. SConsX is an
293 extension package of SCons (Knight, 2005). It simplifies the building of platform
294 dependent packages by supporting different types of compilers (i.e. GCC, MinGW, Visual
295 C++) and platform environments. Similarly, Deploy extends the standard Setuptools library
296 for packaging and installation of modules by adding a support for reusable components
297 with shared libraries. A graphical front-end of this tool has been developed to facilitate the
298 install, update or removal of OpenAlea packages on Windows, Linux and MacOS X
299 platforms. The user selects the packages (s)he needs from a list of available packages. The
300 selected packages and their dependencies are automatically downloaded and installed on
301 the system. The list of available packages is retrieved from standard or user-defined web
302 repositories (e.g. OpenAlea GForge public web repository or personal private repository
303 using authentication). Third-party Python packages of the Python Package Index (PyPI,
304 <http://pypi.python.org>) are also accessible through this interface.

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

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

326 *Currently integrated components*

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

330

- 331 • Plant architecture analysis: the VPlants package, successor of AMAPmod, provides
332 data structure and algorithms to store, represent and explore multi-scale plant
333 architectures. Statistical models like Hidden-Markov tree models (Durand *et al.*
334 2007) or change points detection models (Guédon *et al.* 2007) are provided to
335 analyze branching pattern and tree architecture.
- 336 • Plant geometry modeling: The PlantGL graphic library (Pradal *et al.* 2007) contains
337 a hierarchy of geometric objects dedicated to plant representations that can be

338 assembled into a scene graph, a set of algorithms to manipulate them and some
339 visualization tools. Some parametric generative processes to build plant architecture
340 (e.g. Weber and Penn 1995) are also integrated.

341 • Eco-physiological processes: Caribu (Chelle and Andrieu 1998) and RATP
342 (Sinoquet *et al.* 2001) provide methods for light simulation in 3D environments and
343 for computing radiation interception, transpiration, and carbon gain of a tree
344 canopy. The Drop model (Dufour-Kowalski *et al.* 2007) simulates rainfall
345 interception and distribution by plants.

346 • Meristem modeling: Mechanical models of tissue compute cell deformation and
347 growth (Chopard *et al.* 2007).

348 • Finally, a catalog component provides common tools for general purposes such as
349 simple mathematical functions, standard data structures (e.g. string, list, dictionary,
350 *etc.*), and file manipulation services.

351 **A case-study of use of OpenAlea in ecophysiology: estimation by simulation of light** 352 **interception efficiency**

353 *Overview*

354 The objective, in this case-study, was to determine how the integral of the fraction of light
355 intercepted by a maize crop over the plant cycle is sensitive to natural variation in leaf
356 shapes. To do so, the light interception efficiency (LIE) is estimated by a simulation
357 procedure using different leaf shapes which were measured in the field for a given number
358 of maize genotypes. This procedure required the use of three types of model: (i) a model of
359 3D leaf shapes, (ii) a simulator of the development of the canopy, here ADEL-maize

360 (Fournier and Andrieu, 1998), and (iii) a radiative model, here Canestra (Chelle and
361 Andrieu, 1998).

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

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

371

372 *From field data to 3D leaf shapes*

373 Two properties of leaf shapes were measured: the variation of leaf width as a function of
374 the distance from the base of the leaf, and the 3D trajectory of the leaf midribs. In previous
375 uses of ADEL-maize, an analytical model of leaf shape, i.e. composed of conic arcs (Prevot
376 *et al.* 1991), was fitted to the data to smooth them out and remove digitizing errors. The
377 estimated parameters of this leaf model were used as inputs to the L-system based 3D plant
378 generator. In this case-study, we have developed a new parametric model because the shape
379 of midrib leaf curves of certain genotypes presents several inflexion points which can not
380 be easily approximated using conics. This was not done before due to the difficulty to
381 design new algorithm which used external scientific libraries. The midrib curve and the
382 variation of the leaf width are approximated, in the parametric model, with NURBS curves

383 using the least square fitting algorithm (Piegl and Tiller 1997), available in the Python
384 scientific library, SciPy (Oliphant 2007). To optimize the final radiative computation,
385 whose complexity depends on the square of the number of triangles of the leaves, the
386 NURBS curves have been simplified as polylines with a given number of points using a
387 decimation algorithm (Agarwal and Varadarajan 2000) developed in Python. Under
388 VisuAlea (Figure 5.A), the user can graphically set the leaf data and control the level of
389 discretization of the final mesh by setting the values of the ‘fit leaves’ nodes which convert
390 the leaf measurement into simplified polylines. Using knowledge about maize leaf
391 development (Fournier and Andrieu, 1998), the leaf shape can be reconstructed at any stage
392 of its development. To obtain the leaf shape from the curves and user-defined
393 developmental parameters (e.g. length, radius, ...), a PlantGL mesh is computed by
394 sweeping a section line of length following the width variation along the approximated
395 midrib curve. Such reconstruction was handled by the ‘symbols’ node (Figure 5.A.4) and
396 used during the geometric reconstruction of the plant.

397 *From 3D leaf shapes to canopy development*

398 In previous applications, ADEL-maize, which is a cpfg script, was used to simulate directly
399 canopy 3D development. The simulation was done in two steps. First, the model computed
400 the evolution of the topology and of the dimensions of the organs of each plant, and stored
401 it as a string. Second, a 3D mockup of the canopy was computed using the cpfg interpreter
402 and a homomorphism. In this application, we did not apply the homomorphism to be able
403 to use the geometric leaf shapes built outside cpfg. The plant reconstruction was performed
404 from the L-system string using LOGO style turtle interpretation (Prusinkiewicz, 86)

405 implemented in PlantGL (Pradal *et al.*, 2007). Finally, the resulting individual plant mock-
406 ups were sent to a planter node that distributed the plants over a defined area.

407

408 *From Canopy reconstruction to LIE*

409 LIE was computed with the radiative model Caribu, which is a package of OpenAlea. The
410 model is itself composed of several programs that can be arranged to fit particular needs.
411 We used one of the arrangements that computes first order interception for an overcast sky,
412 issued in the package in the form of a VisuAlea dataflow. We simply saved this Caribu
413 dataflow as a composite node, imported it to the Adel dataflow (see Figure 5.A), and made
414 connections between slots. This package also already included visualization tools based on
415 PlantGL (such as the one producing output in Figure 5.C) and post-treatment routines for
416 computing LIE. The complete dataflow (Figure 5.A) could be saved as a composite node
417 and used in a new dataflow that iterates on different input datasets (similarly to Figure 4).

418

419 In this application, OpenAlea was used to extend the capabilities of the original application
420 and to re-implement it in a more modular way, while improving the clarity of the chaining
421 of the models. The ADEL application has inherited new features from the use of already
422 existing tools. These new features include a) a parametric model to represent leaf shapes
423 using parametric surfaces computed directly from digitized leaves; b) user control of the
424 number of polygons used to represent leaf shapes, and c) access to a large palette of sowing
425 strategies. Visualization and plotting tools are provided by PlantGL to generate different
426 kinds of outputs (images, animations ...). Although the dataflow presented in Figure 5.A is
427 specific to this particular application, it is easily editable and configurable for other

428 objectives. For example, we can easily imagine replacing the maize model by another plant
429 model, even developed with another simulator. All this finally requires a very limited
430 programming effort, thanks to the re-use of libraries, and the automatic generation of
431 graphical interfaces under VisuAlea.

432

433

434 **Conclusion**

435

436 The major achievement of OpenAlea is to provide a visual and interactive interface to the
437 inner structure of an FSPM application. This greatly improves the potential of sharing and
438 reusing specialized integrated models, since embedded sub-models, data-structures, or
439 algorithms can be recomposed or combined to fit different modeling objectives. This also
440 increases, for end users, the knowledge of how an application works as one can evaluate
441 independently any part of the model dataflow. As OpenAlea is primarily intended for the
442 FSPM community, we hope that such a platform will facilitate the emergence and sharing
443 of generic components and algorithms able to perform standard modeling tasks in this
444 domain. We also paid a particular attention to providing tools to ease the integration of
445 existing models, so that a large community of scientists could use and “feed” the platform.
446 In its present state, OpenAlea is suited to build examples like the one presented here, where
447 individual components have to be chained sequentially, and with a genericity of algorithms
448 at the level of model subunit. The visual programming environment has been designed for
449 models integration and connection rather than for modeling feedback and retroaction
450 between models. It has been based on a dataflow model of computation where control flow
451 and feedback are difficult to represent, like in functional languages. However, retro-action

452 and feedback can be managed within specific nodes like simulation nodes or biophysical
453 solvers. OpenAlea only partially addresses the question, pointed by Prusinkiewicz et al.
454 (2007), regarding the construction of comprehensive models that incorporate several
455 aspects of plant functioning with intricate interactions between functions (for example a
456 plant development model coupled with hormonal control, partitioning of resources, water
457 fluxes and biomechanics). This would probably require to define and share generic data
458 structures representing the plant on different scales, and address, both theoretically and
459 algorithmically, the problem of simulating different processes acting in parallel on different
460 scales.

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

465

466 **Acknowledgments**

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

471

472 **References**

- 473 Agarwal P K, Varadarajan K R (2000) Efficient algorithms for approximating polygonal
474 chains. *Discrete and Computational Geometry* **23**, 273-291
- 475 Ascher D, Lutz M (1999) Learning Python. *Sebastopol, California: O'Reilly and*
476 *Associates.*
- 477 Chelle M, Andrieu B, (1998) The nested radiosity model for the distribution of light within
478 plant canopies. *Ecological Modelling* **111**, 75-91
- 479 Chopard J, Godin C, Traas J (2007) Toward a formal expression of morphogenesis: a
480 mechanics based integration of cell growth at tissue scale. *Proceedings of the 7th*
481 *International Workshop on Information Processing in Cell And Tissues (IPCAT 2007)*,
482 388-399
- 483 Councill B, Heineman G T (2001) Definition of a software component and its elements.
484 *Component-based software engineering: putting the pieces together, Addison-Wesley*
485 *Longman Publishing Co., Inc.*, 5-19
- 486 Demsar J, Zupan B, Leban G (2004) Orange: From Experimental Machine Learning to
487 Interactive Data Mining. *White Paper, Faculty of Computer and Information Science,*
488 *University of Ljubljana.*
- 489 Dufour-Kowalski S, Bassette C, Bussière F (2007) A software for the simulation of
490 rainfall distribution on 3D plant architecture: PyDrop. *Proceeding of the 5th*
491 *international workshop on functional structural plant models*, 29(1-3)
- 492 Durand J B, Caraglio Y, Heuret P, Nicolini E (2007) Segmentation-based approaches for
493 characterising plant architecture and assessing its plasticity at different scales.

494 *Proceeding of the 5th international workshop on functional structural plant models,*
495 39(1-3)

496 Evers J B, Vos J, Chelle M, Andrieu B, Fournier C, Struik P C (2007) Simulating the
497 effects of localized red: far-red ratio on tillering in spring wheat (*Triticum aestivum*)
498 using a three-dimensional virtual plant model. *New Phytologist* **176**, 325-336

499 Federl P, Prusinkiewicz P (1999) Virtual laboratory: an interactive software environment
500 for computer graphics. *Computer Graphics International, 1999. Proceedings*, 93-100

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

504 Fournier C, Andrieu B (1998) A 3D Architectural and Process-based Model of Maize
505 Development. *Annals of Botany* **81**, 233-250

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

508 Godin C, Costes E and Sinoquet H (1999) A method for describing plant architecture
509 which integrates topology and geometry. *Annals of Botany* **84**, 343-357

510 Godin C, Sinoquet H (2005) Functional-Structural Plant modelling. *New Phytologist* **166**,
511 705-708

512 Goreaud F, Alvarez I, Courbaud B, de Coligny F (2006) Long-Term Influence of the
513 Spatial Structure of an Initial State on the Dynamics of a Forest Growth Model: A
514 Simulation Study Using the Capsis Platform. *Simulation* **82**, 475-495

- 515 Guédon Y, Caraglio Y, Heuret P, Lebarbier E, Meredieu C (2007) Identifying and
516 characterizing the ontogenetic component in tree development. *Proceeding of the 5th*
517 *international workshop on functional structural plant models*, 38(1-5)
- 518 Higham D J, Higham N J (2005) MATLAB Guide *SIAM: Society for Industrial and*
519 *Applied Mathematic*.
- 520 Johnston W M, Hanna J R P, Millar R J (2004) Advances in dataflow programming
521 languages. *ACM Computing Surveys (CSUR)***36**, 1-34
- 522 Knight S (2005) Building software with Scons. *Computing in Science and Engineering* **7**,
523 79-88
- 524 Kniemeyer O, Buck-Sorlin G, Kurth W (2006) GroIMP as a platform for functional-
525 structural modelling of plants. In: *Functional-Structural plant modelling in crop*
526 *production*, 43-52
- 527 Ludascher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee E A, Tao J, Zhao Y
528 (2006) Scientific workflow management and the Kepler system. *Concurrency and*
529 *Computation: Practice and Experience* **18**, 1039-1065
- 530 Mech R, Prusinkiewicz P (1996) Visual models of plants interacting with their
531 environment. Rushmeier, H. (ed.) *Addison-Wesley*, 397-410
- 532 Oliphant T E (2007) Python for Scientific Computing. *Computing in Science and*
533 *Engineering* **9**, 10-20
- 534 Ousterhout J K (1998) Scripting: Higher-Level Programming for the 21st Century.
535 *Computer, IEEE Computer Society* **31**, 23-30

- 536 Perttunen J, Sievänen R, Nikinmaa E, Salminen H, Saarenmaa H (1996) LIGNUM: A Tree
537 Model Based on Simple Structural Units. *Annals of Botany* **77**, 87-98
- 538 Piegl L, Tiller W (1997) The Nurbs Book. Springer.
- 539 Pommel B, Sohbi Y, Andrieu B (2001) Use of virtual 3D maize canopies to assess the
540 effect of plot heterogeneity on radiation interception. *Agricultural and Forest*
541 *Meteorology* **110**, 55-67
- 542 Pradal C, Boudon F, Nouguier C, Chopard J, Godin C (2007) PlantGL: a Python-based
543 geometric library for 3D plant modelling at different scales. *INRIA Research Report*.
- 544 Prusinkiewicz P (2004) Art and Science for Life: Designing and Growing Virtual Plants
545 with L- systems. *Acta Horticulturae* **630**, 15-28
- 546 Prusinkiewicz P., Karkowski R. & Lane B. (2007) The L+C plant-modelling language. In:
547 *Functional-Structural plant modelling in crop production*, 27-42
- 548 Prusinkiewicz P (1986) Graphical applications of L-systems. *Proceedings of Graphics*
549 *Interface '86*, 247-253
- 550 Prusinkiewicz P, Lindenmayer A (1990) The algorithmic beauty of plants. *Springer-*
551 *Verlag New York, Inc. New York, NY, USA*.
- 552 Prusinkiewicz P, Hanan J (2007) *Proceedings of the 4th International Workshop on*
553 *Functional-Structural Plant Models, FSPM05*.
- 554 Prévot L, Aries F, Monestiez P (1991) Modélisation de la structure géométrique du maïs.
555 *Agronomie* **11**, 491-503

556 R Development Core Team (2007) R: A Language and Environment for Statistical
557 Computing.

558 Raymond E S (2004) The Art of Unix Programming. *Addison-Wesley Professional*.

559 Sanner M F (1999) Python: a programming language for software integration and
560 development. *Journal of Molecular Graphics and Modelling* **17**, 57-61

561 Sanner M F, Stoffler D, Olson A J (2002) ViPEr, a Visual Programming Environment for
562 Python. *Proceedings of the 10th International Python Conference* 103-115

563 Da Silva D, Boudon F, Godin C, Puech O, Smith C, Sinoquet H (2006) A Critical
564 Appraisal of the Box Counting Method to Assess the Fractal Dimension of Tree
565 Crowns. *Lecture Notes in Computer Sciences* **4291**, 751-760

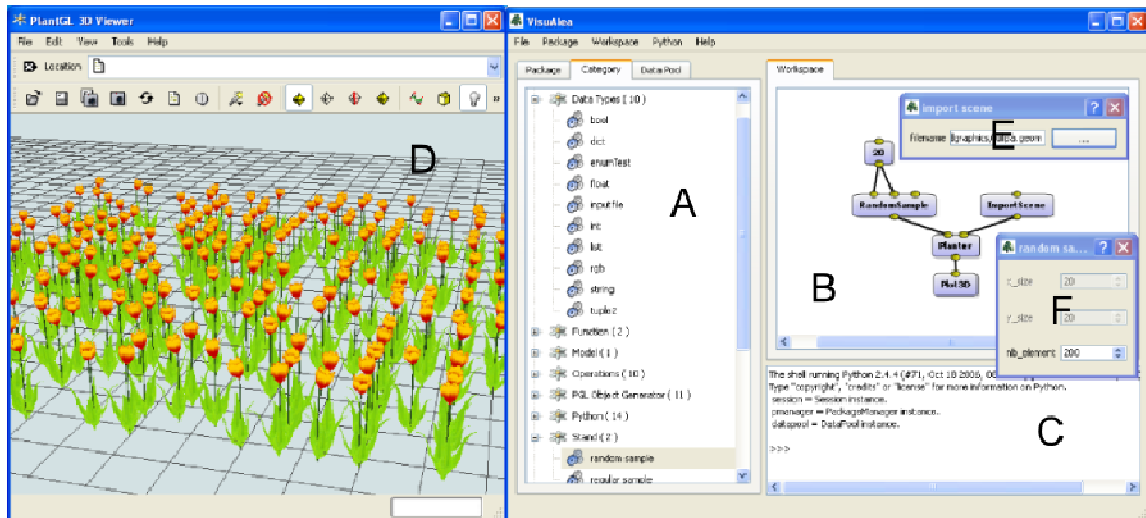
566 Sinoquet H, Roux X L, Adam B, Ameglio T, Daudet F A (2001) RATP: a model for
567 simulating the spatial distribution of radiation absorption, transpiration and
568 photosynthesis within canopies: application to an isolated tree crown. *Plant Cell and*
569 *Environment* **24**, 395-406

570 Szyperski C (1998) Component Software: Beyond Object-oriented Programming.
571 *Addison-Wesley Professional*.

572 Tardieu F (2003) Virtual plants: modelling as a tool for the genomics of tolerance to water
573 deficit. *Trends in Plant Science* **8**, 9-14

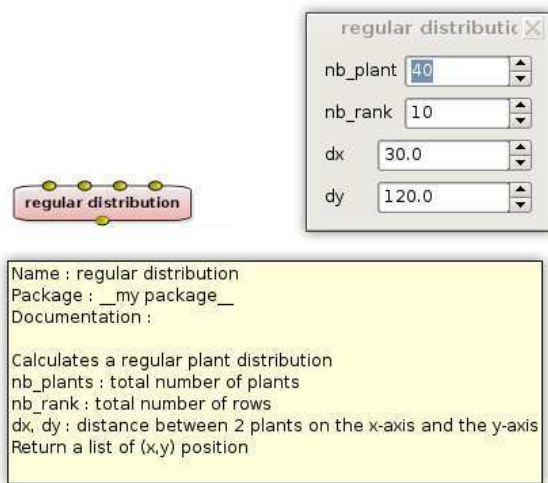
574 Upson C, Faulhaber T A, Kamins D, Laidlaw D, Schlegel D, Vroom J, Gurwitz R, van
575 Dam A (1989) The application visualization system: a computational environment for
576 scientific visualization. *Computer Graphics and Applications* **9**, 30-42

- 577 Vos J, Marcelis L F M, De Visser P H B, Struik P C, Evers J B (2007) Functional
578 structural plant modelling in crop production. *Springer, The Netherlands,*
- 579 Weber J, Penn J (1995) Creation and rendering of realistic trees. Proceedings of the 22nd
580 annual conference on Computer graphics and interactive techniques, 119-128



581
 582 Figure 1. Snapshot of the OpenAlea visual modeling environment. (A) The package
 583 manager list packages and nodes found on the system. (B) The graphical programming
 584 interface enables users to build visual dataflow by interconnecting nodes. A 3D scene is
 585 built by associating a single geometry with a random distribution of points. (C) Low level
 586 interactions are done in the Python interpreter. (D) A 3D viewer is directly called by the
 587 Plot3D component. (E-F) Widgets specific to each component are automatically generated.

588



The image shows a graphical node for a function named 'regular distribution'. The node is represented by a pink rounded rectangle with the text 'regular distribution' and five yellow circles above it. A tooltip is displayed below the node, containing the following text: 'Name : regular distribution', 'Package : __my package__', 'Documentation :', 'Calculates a regular plant distribution', 'nb_plants : total number of plants', 'nb_rank : total number of rows', 'dx, dy : distance between 2 plants on the x-axis and the y-axis', and 'Return a list of (x,y) position'. To the right of the node is a control panel titled 'regular distributic' with four input fields: 'nb_plant' (value 40), 'nb_rank' (value 10), 'dx' (value 30.0), and 'dy' (value 120.0). Each field has a small up/down arrow on its right side.

```
def regular(nb_plant, nb_rank, dx, dy):  
    """  
    Calculates a regular plant distribution  
    nb_plants : total number of plants  
    nb_rank : total number of rows  
    dx, dy : distance between 2 plants on the x-axis and the y-axis  
    Return a list of (x,y) position  
    """  
    nx = int( nb_plant / nb_rank )  
    ny = nb_rank  
  
    return [ ( i * dx, j * dy )  
            for j in xrange(nb_rank)  
            for i in xrange(nx)  
            ],
```

589

590

591 Figure 2. A graphical node is a visual representation of a function. Input ports at the top

592 represent the input arguments and output ports at the bottom, the resulting values. In this

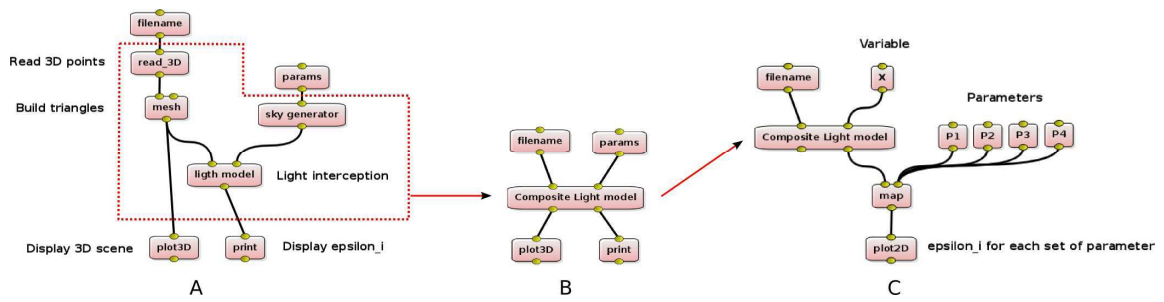
593 example, the "regular" node generates a list of position (x,y) corresponding to a regular

594 plant distribution. Documentation is automatically extracted and display in a tooltip. The

595 node widget allows to set the value of the parameters. On the right, we show the related

596 Python code.

597



598

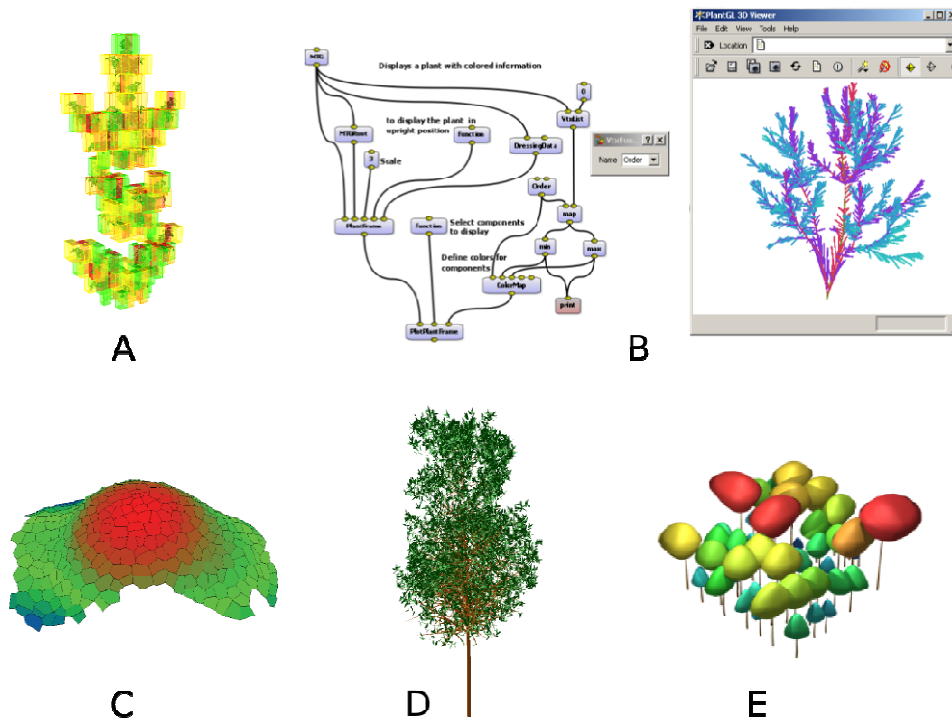
599 Figure 3. In the first example, we construct a plant model from a set of 3D points read in a
 600 file. Then, the light interception is computed using a sky description. The 3D plant is

601 displayed in a 3D viewer, and the results of the light model are displayed in the shell. In the

602 second example, the dataflow is simplified by grouping some nodes in a composite node.

603 The third example shows the same model applied for different set of parameters.

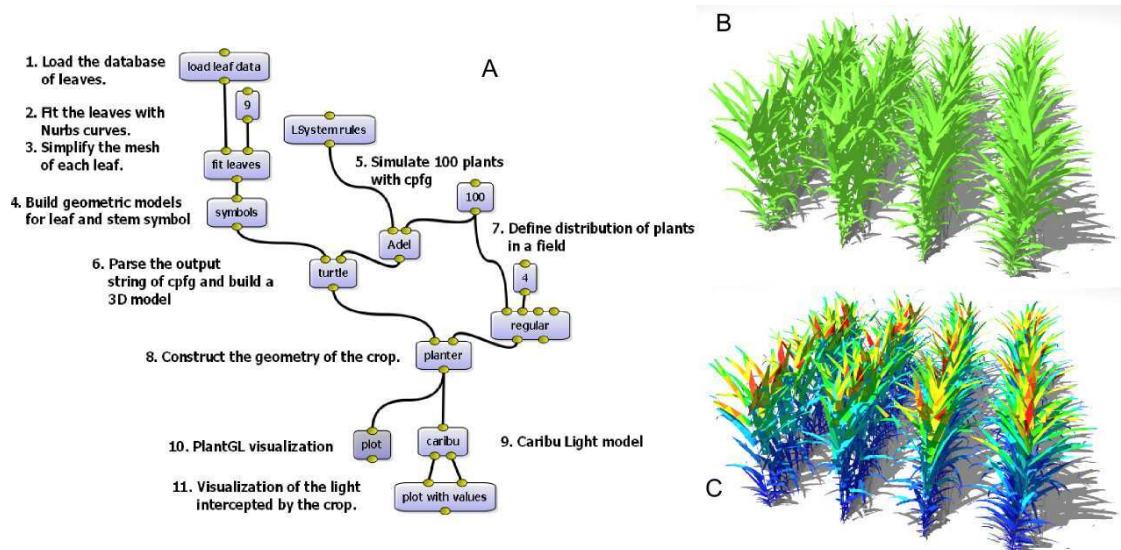
604



606

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

613



614

615 Figure 5. Snapshots of the VisuAlea dataflow (A), and of two outputs of an application
 616 allowing to reconstruct a maize canopy (B) and to estimate light distribution within it (C).

617 Annotations on the dataflow succinctly describe the functions of the different nodes. Nodes

618 1 to 4 defines the leaf shape model, which is a function that returns leaf shape at a given
 619 stage of development, from a set of curves fitted to digitize mature leaf shape data. Node 5

620 is an L-System engine simulating plant development from an L-system script
 621 ('LSystemRules'). Nodes 6 to 8 are for the reconstruction of the 3D scene: one node

622 combines the L-system output with the leaf model to reconstruct the plants ('turtle'), and
 623 one node ('planter') is used for placing plants according to a pattern ('regular'). Node 9 is

624 for the radiative model, and node 10 and 11 are for producing 3D outputs (B and C). Three
 625 parameters are represented with nodes to allow a direct interaction with the application: the

626 number of polygons used to represent leaves (9), the total number of plants in the scene
 627 (100) and the number of rows (4).

628