Architecture de dataflow pour des systèmes modulaires et génériques de simulation de plante

Christophe Pradal

THESE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

PAR LA VALIDATION DES ACQUIS DE L'EXPERIENCE

Année universitaire 2018-2019

En Informatique

École doctorale I2S – Information, Structures, Systèmes

Unité de recherche UMR AGAP – Université de Montpellier

SYNTHÈSE DES TRAVAUX DE RECHERCHE

Architecture de Dataflow pour des systèmes modulaires et génériques de simulation de plante

Présentée par Christophe PRADAL

Le 17 juillet 2019

Référent : Christophe GODIN

Devant le jury composé de

Christophe GODIN, Directeur de recherche, Inria, Lyon	Référent
Marianne HUCHARD, Professeur des Universités, Université de Montpellier	Représentante de l'École Doctorale
Jean-Louis GIAVITTO, Directeur de recherche, CNRS, Ircam, Paris	Rapporteur
Gerhard BUCK-SORLIN, Professeur, AGROCAMPUS OUEST, INHP, Angers	Rapporteur
Sarah COHEN-BOULAKIA, Professeure des universités, Université Paris-Saclay	Examinatrice
Dennis SHASHA, Professor, New York University	Examinateur
Patrick Valduriez, Directeur de recherche, Inria, Montpellier	Examinateur
Bertrand Muller, Directeur de recherche, INRA, Montpellier	Examinateur

UNIVERSITÉ DE MONTPELLIER

Table des matières

Chapitre 1

Introduction

Les modèles de plantes structure-fonction (nommés aussi Functional-Structural Plant model ou FSPM) sont devenus un objet d'étude à part entière dans les années 90. Ils ont été développés pour comprendre la relation complexe entre l'architecture de la plante et les processus biologiques et physiques qui influencent la croissance (Godin, Costes et Sinoquet, 2005). Ils se fondent sur une représentation explicite de la morphologie des plantes et sur une modélisation mécaniste des processus écophysiologistes et bio-physiques du fonctionnement de la plante (Prusinkiewicz et Lindenmayer, 1990 ; Sievanen, Makela et Nikinmaa, 1997 ; Godin, Costes et Sinoquet, 2005 ; Hanan et Prusinkiewicz, 2008 ; Fourcaud et al., 2008 ; Vos et al., 2010).

Les échelles considérées par ces modèles sont multiples, allant de l'échelle de la cellule, des tissus, de l'organe, de l'organisme, jusqu'au peuplement (Godin et Caraglio, 1998 ; Godin, Costes et Sinoquet, 2005 ; Bucksch et al., 2017 ; Balduzzi et al., 2017 ; Marshall-Colon et al., 2017a). De plus, une des propriétés remarquables des systèmes biologiques est leur organisation hiérarchique (Jacob, 1987 ; Lucas, Laplaze et Bennett, 2011). Les plantes sont composées d'organes, qui sont constitués de tissus, eux-mêmes contenant des cellules, composées d'organelles, qui sont formées de macro-molécules et ainsi de suite. L'étude de ces systèmes complexes devient de première importance en biologie pour prendre en compte la morphologie dans l'étude des interactions entre les éléments des systèmes biologiques, ainsi qu'avec l'environnement (Godin, Costes et Sinoquet, 2005 ; Lucas, Laplaze et Bennett, 2011 ; Vos et al., 2010 ; Bucksch et al., 2017). L'étude de la morphologie des plantes se retrouve à l'interface de nombreuses disciplines biologiques. En écologie végétale, par exemple, la morphologie des communautés définit le type de végétation et le biome, incluant leur relation à l'environnement. A contrario, la morphologie des plantes est modulée par le fonctionnement des plantes, étudié en physiologie végétale. Elle est aussi impactée par les réseaux de gènes et la synthèse des protéines, étudiés en biologie moléculaire (Kaplan, 2001).

Mais la morphologie est plus qu'un attribut contrôlant l'organisation structurelle des plantes. La plante est un organisme en croissance, qui se développe dans l'espace et le temps (Barthélémy et Caraglio, 2007). Le développement des plantes est programmé génétiquement et piloté par des processus biochimiques. De plus, les forces externes modulent le développement de la plante, comme l'hétérogénéité de la densité du sol qui affecte la croissance racinaire, ainsi que les flux d'air, d'eau ou la gravité qui modifient la courbure des branches et des feuilles (Moulia et Fournier, 2009).

Les systèmes biologiques végétaux sont donc des systèmes complexes de grande taille, dynamiques, hiérarchiques et très structurés qui ne peuvent plus être étudiés uniquement par des approches directes, où chaque composante du système peut-être isolée et son comportement observé par une approche expérimentale. La modélisation est un moyen pour comprendre leurs structures et leurs comportements (Lucas, Laplaze et Bennett, 2011).

Ces dernières années, de nombreux formalismes ont été proposés pour la représentation, la modélisation et la simulation de ces systèmes. Ces formalismes sont issus de différentes disciplines, comme les mathématiques, l'informatique, la physique ou la biologie. Leur hétérogénéité montre la grande variété des points de vues que peuvent avoir des chercheurs de différentes disciplines.

En mathématique et en informatique, plusieurs formalismes ont été proposés pour décrire la forme et la morphologie des plantes. Plusieurs représentations mathématiques existent, comme les graphes, pour représenter la structure de la plante. En particulier, les formalismes les plus utilisés sont les arborescences, les arborescences quotientées, les arborescences multi-échelles, les graphes (Godin, 2000) ainsi que les graphes multi-échelles (Ong et al., 2014). D'autres représentations plus spécifiques existent cependant pour représenter la topologie ou la géométrie des plantes comme les séquences (Guédon et al., 2001), les surfaces ou les volumes (Pradal et al., 2009). Le choix de la représentation dépend grandement de l'application visée mais aussi de contraintes liées à la mesure, par exemple.

Pour l'analyse et la recherche de motifs dans ces structures végétales, des méthodes de nature combinatoire (Ferraro, Godin et Prusinkiewicz, 2005), statistique (Guédon et Costes, 1998 ; Guédon, Barthélémy et Caraglio, 1999 ; Guédon et al., 2001 ; Guédon, Heuret et Costes, 2003 ; Guédon et al., 2007) ou topologique (Li et al., 2017) ont été utilisées. De façon similaire, différents formalismes ont été utilisés pour caractériser la géométrie des plantes, comme l'analyse fractale (Oppelt et al., 2000 ; Halley et al., 2004 ; Boudon et al., 2006 ; Da Silva et al., 2006, ou l'analyse spectrale (Kendall, 1984 ; Mebatsion et al., 2011 ; Chitwood et al., 2012).

De même, pour simuler la croissance des plantes, différents formalismes informatiques ont été utilisés, comme les grammaires formelles et les languages de réécriture (Lindenmayer-Systems Prusinkiewicz et Lindenmayer, 1990, Relational Growth Grammar Kurth, Kniemeyerand et Buck-Sorlin, 2005), les systèmes à évenements discrets (formalisme Discrete Events System Specification Zeigler, Praehofer et Kim, 2000), mais aussi les équations différentielles, très utilisées en épidémiologie, ou encore les équations aux différences, formalisme prépondérant utilisé dans les modèles de culture ou pour simuler des processus de flux (eau, hormones) dans la plante (e.g. Doussan, Pagès et Vercambre, 1998 ; Renton et al., 2012 ; Pallas, Costes et Hanan, 2016).

On observe cette même diversité de formalisme en biologie, pour modéliser les processus qui gouvernent le fonctionnement des plantes. Par exemple, un des processus essentiel pour la croissance et le fonctionnement est l'environnement lumineux perçu par les organes de la plante. Plusieurs modèles permettent de modéliser la lumière absorbée comme le lancer de rayon (ray casting, Monte Carlo Ray Tracing, Quasi Monte Carlo Ray Tracing) ou la radiosité (Chelle et Andrieu, 1999 ; Vos et al., 2010 ; Fournier et al., 2016). Ces approches permettent de calculer la lumière interceptée par une canopée représentée par un ensemble de triangles. D'autres approches calculent le bilan d'énergie sur d'autres représentations, à d'autres échelles. La canopée peut être représentée comme un ensemble de couches infinies (approche 1D), ou comme un ensemble d'éléments volumiques ou voxels. Dans ce cas, le calcul du bilan d'énergie se fait entre éléments de volume poreux (hypothèse milieu turbide) avec une hypothèse d'homogénéité au sein de chaque élément. La géométrie des organes est alors réduite à une densité de feuilles (Sinoquet, Moulia et Bonhomme, 1991 ; Vos et al., 2010).

Aujourd'hui, la société est confrontée à des défis majeurs que sont le changement climatique et la sécurité alimentaire. Il est nécessaire de comprendre le fonctionnement des plantes et les mécanismes qui gouvernent à l'élaboration des formes végétales. Mais il est aussi important de pouvoir tester des scénarios complexes liés à des situations qui ne sont pas ou difficilement observables, comme l'impact d'une augmentation de température sur le développement et la propagation de maladies foliaires ou d'insectes (Saudreau et al., 2013), ou la conduite de vergers par manipulation de la structure des arbres pour optimiser l'interception de la lumière afin d'augmenter la production (Jackson, 1980 ; Fumey et al., 2011).

Modéliser la croissance et le fonctionnement des plantes implique de modéliser un système dynamique à structure dynamique. Les processus fonctionnels à intégrer sont très variés et sont modélisés à différentes échelles. De nombreux modèles ont été

développés, des bases de données existent, mais rares sont les modèles structure-fonction qui intègrent des modèles existants sans avoir à les redévelopper pour pouvoir les intégrer. Cela explique peut-être pourquoi peu de modèles intègrent plus de deux ou trois processus et cela uniquement sur un type de plante (e.g. LPeach (Allen, Prusinkiewicz et DeJong, 2005), MappleT (Costes et al., 2008), Adel (Fournier et Andrieu, 1998)). En effet, l'implémentation d'un processus particulier est souvent dépendante d'un type de plante et n'est pas réutilisable.

Ceci peut-être du au fait que les modèles des processus fonctionnels ne sont pas développés, ni testés indépendamment du type de plante et qu'ils sont souvent dépendants d'une structure de données particulière qui ne permet pas toujours de représenter tous les types de plantes ou tous les organes d'une plante. Cependant, récemment, Henke et coll. (Henke, Kurth et Buck-Sorlin, 2016) ont développé un prototype d'un modèle FSPM (FSPM-P) sur plantes annuelles ou bi-annuelles afin de pouvoir réutiliser facilement des modèles source-puit de carbone en suivant un prototype développé dans la plateforme GroIMP. Mais en dehors d'un tel cadre prédéfini qui fixe l'échelle de representation et les variables d'état des processus, les modèles dépendent souvent implicitement d'une échelle de représentation particulière, ou dépendent explicitement de la présence d'autres processus et communiquer avec eux en modifiant des variables globales. Si toutes les variables dont ils dépendent ne sont pas explicitées et qu'ils en fixent certaines à des valeurs constantes dans le code en fonction de la connaissance particulière d'un type de plante, il faut modifier le modèle pour pouvoir le réutiliser. De la même façon, les modèles peuvent contenir des simplifications qui ne sont valables que pour un groupe de plantes donné.

Il est évidemment souhaitable de pouvoir facilement réutiliser les logiciels et modèles qui ont été développés par des spécialistes dans leurs disciplines respectives. Cela permet de transférer la connaissance d'un domaine de recherche à l'autre, mais aussi de bénéficier du travail effectué pour tester le modèle et pour le valider. Cette réutilisation des modèles et des logiciels est d'autant plus cruciale que le problème est complexe. Il fait intervenir différents processus à différentes échelles et nécessite l'intégration de sources variées de données provenant de divers capteurs. La communauté scientifique en biologie des plantes prend conscience aujourd'hui des fortes interactions, et ce à différentes échelles, entre le développement de l'architecture et l'environnement (Godin, Costes et Sinoquet, 2005 ; Saudreau et al., 2013 ; Marshall-Colon et al., 2017a ; Tardieu et al., 2017).

Cette prise de conscience s'est faite bien plus tôt en informatique. C'est en 1968, lors de la conférence NATO en Génie Logiciel, que le terme *software crisis* ou *crise logicielle* est apparu (Buxton et Randell, 1970 ; Krueger, 1992). La crise logicielle

dénote l'incapacité de construire de grands systèmes logiciels fiables de façon maîtrisée et en en contrôlant les coûts. Lors de cette conférence, dévolue à cette question et considérée comme le lieu de naissance du domaine du Génie Logiciel, la réutilisation logicielle a été vue comme un moyen d'éviter cette crise (McIlroy et al., 1968). Aujourd'hui encore, de nombreux informaticiens voient la réutilisation logicielle comme un moyen d'améliorer les pratiques en développement logiciel et d'amortir les efforts de développement même si les pratiques, méthodes et langages ont changé significativement ces cinquante dernières années.

C'est à la question de la modularité et de la réutilisation logicielle dans les modèles *FSPMs* que cette thèse va tenter d'apporter une réponse. Dans cette introduction, nous allons d'abord présenter la taxonomie des différentes stratégies de réutilisation logicielle proposée par Krueger. Ensuite, nous décriront les architectures à tableau noir qui permettent la coopération d'artefacts logiciels, puis les systèmes de workflows scientifiques qui abordent un autre aspect de la réutilisation de composants logiciels. Finalement, nous discuteront des limites des approches actuelles.

1.1 Etat de l'art des réponses

1.1.1 La réutilisation logicielle vue par Krueger

La réutilisation logicielle est l'isolation, la sélection, la maintenance et l'utilisation d'artefacts logiciels existants dans le développement d'un nouveau système (Reese et Wyatt, 1987). On parle d'artefact logiciel car la réutilisation concerne aussi bien des fragments de code source, que des architectures logicielles, des bibliothèques, des spécifications, de la documentation ou des approches comme la génération de code (Freeman, 1983).

Krueger a défini une taxonomie caractérisant les différentes approches et la façon dont les artefacts logiciels sont abstraits, sélectionnés, spécialisés et intégrés (Krueger, 1992).

Sa taxonomie est composée de huit stratégies de réutilisation logicielle illustrant les différentes catégories d'approches proposées dans la littérature : les langages de haut-niveau, la récupération de code, les composants logiciels, les schémas logiciels, les générateurs d'application, les langages de très-haut niveau, les systèmes de transformation et les architectures logicielles.

Les stratégies définies par Krueger sont les suivantes :

— **Les langages de haut-niveau**. Les langages de programmation, comme C, C++, Java, ou Haskel par exemple, mais aussi d'autres langages comme Python ou R, ne sont pas souvent présentés dans la littérature comme des exemples de réutilisation logicielle en tant que tel. Cependant, ils offrent aux développeurs les mêmes avantages que ce que promet la réutilisation logicielle : produire du code plus vite, plus efficace et plus simplement. Comparés à l'assembleur, ici considéré comme langage de bas-niveau, les langages compilés et interprétés fournissent des constructions efficaces pour exprimer le calcul. Les motifs de programmation qui étaient utilisés en assembleur sont devenus des structures de base de ces langages. Les exemples en sont par exemple l'itération, le branchement, les expressions arithmétiques, la déclaration de données ou l'assignation. Chaque famille de langage apporte de nouveaux concepts et structures (e.g. les objets, la gestion automatique de la mémoire, la spécialisation) qui permet d'exprimer simplement des motifs qui seront traduits automatiquement en du code machine, par un compilateur par exemple. L'abstraction utilisée par le développeur lui permet ainsi de construire à la fois des systèmes plus complexes, plus fiable et de les contruire plus rapidement. Les langages de programmation de haut-niveau permettent de produire du code beaucoup plus rapidement que dans des langages bas-niveau car i) les constructions des langages sont succinctes et naturelles, ii) le compilateur ou l'interpréteur convertit automatiquement les structures du langage en code machine et iii) les développeurs sont complètement isolés des détails du compilateur et du code machine.

  Cependant, la principale limitation de cette technologie est que les langages de programmation sont des abstractions de bas-niveau. Il y a une grande distance cognitive entre la conception d'un modèle ou d'un système logiciel d'une part et son implémentation dans un langage de programmation d'autre part. En effet, dans le cycle de développement d'un système ou d'un modèle, l'implémentation vient après la phase de conception.

— **La récupération de code**. La récupération de code est une technique à la fois simple et efficace, ce qui en fait son succès chez les développeurs. Elle consiste à réutiliser de façon ad-hoc, souvent par du copier-coller, des fragments de code ou d'architecture pour construire un nouveau système à partir de systèmes existants. L'objectif de cette technique est de réduire le temps pour construire un nouveau système. Dans l'idéal, un développeur est capable d'adapter de larges fragments de code de grande qualité sans grandes modifications. Dans ce cas, la distance cognitive est faible (copier-coller). Cependant, le risque pour le développeur est de passer plus de temps à localiser,

comprendre, adapter, débugger une partie de code récupérée que de temps à développer son propre code. Cette pratique tend à se généraliser avec des sites spécialisés de recherche de code comme koders ou de partage de code comme stackoverflow, ainsi que du fait du nombre croissant de logiciels libres facilement accessibles sur des sites comme github (`http://github.com`), par exemple.

— **Les composants logiciels**. Les composants logiciels servent de briques de construction qui sont assemblées pour former de plus grands systèmes ou modèles. La nature d'un composant logiciel dépend grandement du langage ou de la technologie dans lequel il est implémenté. Il se décline en une large gamme d'artefacts qui va de la bibliothèque de fonctions (fonctions mathématiques ou d'algèbre linéaire par exemple), à des bibliothèques orientées-objets ou de classes abstraites (la bibliothèque C++ Boost) et à des formes plus sophistiquées de composants comme les services web. Il est à noter que de nombreuses bibliothèques ont été intégrées comme standard dans les langages (Java, Python ou en R). Une différence principale avec d'autres formes de réutilisation est que l'on peut spécialiser un composant sans en modifier le code, uniquement en modifiant ces paramètres. On retrouve cela en C++, dans la spécialisation des classes *template* par exemple, ou dans l'héritage de classe en langage orienté-objet.

  Les composants sont écrits, testés, documentés et stockés spécifiquement pour pouvoir être réutilisés. Certains domaines d'application, comme l'analyse numérique, l'analyse d'image, l'analyse statistique, les bases de données ou la visualisation sont particulièrement bien adaptés aux techniques de réutilisation des composants logiciels, du fait qu'ils disposent d'abstractions bien définies pour les composants (e.g. tableaux multi-dimensionnels, séquences, images, graphes de scène). Dans ces cas, la distance cognitive qui sépare le domaine d'utilisation de l'abstraction offerte par le composant est faible.

  Cependant, les composants sans abstraction bien définie doivent être décrits en langage naturel ou dans un langage décrivant les spécifications. Cette description est souvent aussi complexe à comprendre que le code source lui-même, et la distance cognitive augmente alors. De plus, les librairies de composants généralistes doivent, pour répondre aux besoins variés des développeurs, être beaucoup plus fournies. Elles sont donc plus complexes à installer et à utiliser.

— **Les schémas logiciels**. L'approche par schéma met l'accent sur la réutilisation d'algorithmes et de structure de données abstraites, ainsi que d'abstractions de plus haut niveau. Contrairement aux composants logiciels, les schémas mettent plus l'accent sur la spécification abstraite des algorithmes et des structures de données. On peut classer, dans cette catégorie, la programmation générique (Stepanov et Lee, 1995 ; Alexandrescu, 2001), les schémas de conception ou *design patterns* (Gamma et al., 1995), les diagrammes UML, mais aussi les workflows scientifiques, qui ont comme sémantique d'exécution les dataflows.

La programmation générique, comme la STL en C++, permet de spécifier des algorithmes génériques à partir d'itérateurs. Ces itérateurs, dont les catégories sont spécifiées de façon précise (input, forward, random access), servent de médiateur entre la structure de données et l'algorithme. Ils permettent d'écrire des algorithmes, dont la complexité est garantie, indépendamment de la topologie de la structure de données sous-jacente.

Les workflows scientifiques, que nous détaillerons par la suite, sont souvent représentés et édités dans des environnements logiciels par des graphes dont les sommets représentent les algorithmes et les arcs, les flux de données. Ainsi, des schémas complexes peuvent être réalisés par assemblage de composants pré-existants. L'ordre d'exécution sera automatiquement calculé à partir de la topologie du schéma et de sa sémantique d'exécution, souvent appelée modèle de calcul. En particulier, des algorithmes d'ordonnancement existent pour exécuter de façon efficace, à partir d'une spécification, un workflow sur des infrastructures distribuées (e.g. HPC, cloud, grille de calcul).

Les spécifications formelles d'un schéma permettent d'utiliser des outils automatiques pour le vérifier ou l'assembler. En s'éloignant du code et en manipulant des structures de plus haut niveau, les schémas peuvent diminuer la distance cognitive entre les besoins de l'application et l'implémentation du système. Cependant, la spécification formelle des schémas peut les rendre difficile à comprendre et à réutiliser. Cela peut réduire grandement leur utilité.

— **Les générateurs d'application**. Les générateurs d'application fonctionnent comme des compilateurs. Les fichiers de spécifications sont automatiquement traduits en programme exécutable. Cependant, ils diffèrent des compilateurs traditionnels au sens où les spécifications d'entrée qu'ils prennent sont souvent des abstractions de très haut-niveau, spécifiques à un domaine d'application particulier. En se spécialisant à un domaine d'application, l'expansion du code, c'est à dire le ratio entre le nombre de lignes en entrée de la spécification comparé au nombre de lignes générées, est bien plus grand que pour les

compilateurs des langages de programmation. Alors que les schémas logiciels permettaient de réutiliser des structures de données et des algorithmes, les générateurs d'application vont au-delà en permettant de réutiliser la conception complète d'un système logiciel. Avec les générateurs d'application, les algorithmes et structures de données sont sélectionnés automatiquement ce qui permet au développeur de se concentrer sur ce *que* doit faire le système plutôt que sur *comment* il le fait. Comme exemples de générateur d'applications, on trouve les grammaires (Lex and Yacc), les expressions régulières, les langages graphiques (Scratch, Yahoo Pipes, ...) et la programmation par templates. Les avantages de cette approche est qu'elle associe automatiquement les abstractions de haut-niveau d'un domaine à une forme exécutable. Cela réduit la distance cognitive. Les inconvénients sont que cette approche est très dépendante d'un domaine d'application, il n'est pas toujours possible de trouver un générateur d'application pour un problème de développement logiciel particulier. Et il est aussi difficile d'implémenter un générateur d'application avec les fonctionnalités et les performances désirées pour une large gamme de problèmes.

— **Les langages de très haut-niveau**. Comme le nom le suggère, l'approche des langages de très haut-niveau est de s'appuyer sur le succès des langages de haut-niveau en cherchant à le prolonger. Les langages de haut-niveau (e.g. C, C++, Java) offrent des constructions de haut-niveau fondées sur des théories et des paradigmes robustes et largement diffusés (λ-calcul pour les langages fonctionnels, paradigme orienté-objets par exemple). Les langages de haut-niveau sont soit compilés en langage assembleur, soit dans une représentation plus haut-niveau pour des langages plus récents (Java, .Net) afin d'assurer leur portabilité. De façon similaire les langages de très haut-niveau permettent aux développeurs de créer des systèmes exécutables à partir de constructions qui sont considérées de plus haut-niveau comparées aux langages de haut-niveau. A la limite entre ces deux mondes se trouve les langages interprétés comme le langage Python ou R, qui peuvent être classés dans les deux catégories, selon les points de vue. Comme langage de très haut-niveau, on trouve les DSL ou *Domain Specific Language*, qui sont moins généralistes que les langages interprétés mais capturent les abstractions adaptées à un domaine particulier. Les langages de très haut-niveau ressemblent aux générateurs d'application dans le sens où les deux approches traduisent automatiquement des spécifications de haut-niveau en systèmes exécutables. Cependant, alors que les générateurs d'application utilisent des abstractions spécifiques à un domaine

d'application, les langages de très haut-niveau utilisent des abstractions indépendantes de l'application et génériques. Dans les générateurs d'application, la généralité est sacrifiée pour pouvoir capturer des spécifications de plus hauts-niveaux d'abstraction. Ainsi, les générateurs d'application ont l'avantage d'être plus succincts et puissants dans un relativement petit nombre de cas, alors que les langages de très haut-niveau ont l'avantage d'être plus généralistes vis à vis du développement logiciel, au détriment des performances.

— **Les systèmes de transformation**. Avec les systèmes de transformation, les logiciels sont développés en deux phases :

1. Le comportement sémantique du système est décrit en utilisant un langage de spécification de haut-niveau.
2. Des transformations sont appliquées automatiquement à la spécification de haut-niveau pour la transformer dans une représentation intermédiaire d'un moindre niveau d'abstraction afin optimiser son exécution sans en changer son comportement sémantique.

Ces deux phases font une distinction claire entre *ce que fait* un système et les problèmes d'implémentation liés à *comment il le fait*. Des exemples de systèmes de transformation sont les langages de réécriture, comme les L-systèmes, très utilisés en modélisaton des plantes. Les L-systèmes sont définis par un axiome, un ensemble de modules et des règles de réécritures. L'axiome, représenté par une chaîne de caractères, défini l'état initial du système. Les modules, souvent représentés par des lettres, décrivent les différents composants du système. Les règles de réécritures sont appliquées pour modifier l'état courant du système. Chaque règle est composée de deux parties, une partie définissant un motif et une autre permettant de transformer ce motif en un autre. Si le motif de la règle est trouvé dans l'état courant de la simulation, la règle est appliquée et transforme une sous-partie de la structure.
La première phase des systèmes de transformation est équivalente à celle des langages de très haut-niveau. Par exemple, les L-systèmes utilisent des langages spécifiques (Domain Specific Language ou DSL) pour faciliter l'écriture des règles de façon déclarative plutôt qu'en utilisant un langage impératif et procédural (Boudon et al., 2012).
Par contre, la deuxième phase de transformation n'est pas présente dans les langages de très haut-niveau. Elle permet de réécrire la structure sous-jacente (la chaîne) en appliquant automatiquement les règles sans que l'utilisateur ait besoin de manipuler explicitement cette structure. Et pour ce faire, un code exécutable est généralement produit dont les performances seraient celles d'une

implémentation dans un langage de haut-niveau, et qui satisfait les spécifications de haut-niveau. Dans les systèmes actuels, le production de code se fait soit dans un langage compilé, soit dans un langage interprété. Par exemple, les langages L+C et XL disponibles respectivement à travers les plateformes L-Studio (Prusinkiewicz et al., 1999) et GroIMP (Hemmerling et al., 2008) transforment le code de simulation respectivement en C++ et en Java et le compilent à la volée. De façon similaire, le logiciel L-Py (Boudon et al., 2012) assemble et transforme les règles en Python, un langage dynamique et interprété, ce qui permet d'éviter la phase de compilation, au détriment d'une performance moindre lors de l'exécution.

Ces systèmes de transformations, complètement automatiques ou pouvant être guidées par le développeur, peuvent être vues comme de la compilation. Mais au lieu de produire du code assembleur, elles produisent automatiquement du code dans des abstractions de plus haut-niveau, utilisant des structures de données et des bibliothèques pour finalement compiler le code généré pour garantir les performances.

On retrouve des systèmes de transformation dans des domaines aussi variés que l'ingénierie dirigée par les modèles (i.e. MDE), les workflows scientifiques, les langages de modélisation sémantiques (e.g. SBML, CellML), ou les systèmes de wrapping de code de très haut-niveau. Par exemple, le langage Cython permet, tout en conservant la sémantique du langage Python, de produire du code C optimisé. Le développeur doit indiquer les parties à optimiser et donner des informations sur les types et les fonctions pour que la transformation soit efficace (Behnel et al., 2011). AutoWIG (Fernique et Pradal, 2018) propose une approche complètement automatique en s'appuyant sur l'infrastructure LLVM (Lattner et Adve, 2004) pour générer automatiquement les wrappers Python d'une bibliothèque C++. En effet, l'infrastructure de compilation LLVM donne accès au graphe de compilation de tout code C++ et permet d'implémenter un système d'introspection. En ajoutant des règles de réécritures, on peut donc automatiquement produire les wrappers Python correspondant, de façon complètement automatique. Cette approche, nommée transpilation, est semblable à celle d'un compilateur, sauf qu'elle génère du code C++/Python plutôt que du code machine, directement exécutable.

Inversement, d'autres systèmes, comme Numba, permettent d'optimiser à la volée du code Python lors de son exécution (i.e. Just In Time compilation). Numba s'appuie aussi sur LLVM, non plus pour générer des wrappers en Python, mais pour transformer le code Python en code C++ optimisé.

Ce type de transformation se développe de plus en plus aujourd'hui pour permettre aux développeurs de franchir la barrière des langages et de bénéficier des différentes infrastructures matérielles qui vont des accélérateurs (GPU) à la distribution des calculs sur des machines multi-cœurs, multi-processeurs, des infrastructures en nuage et des grilles de calcul.

Les systèmes de transformation utilisent des abstractions génériques et de haut-niveau. Ces abstractions peuvent être de plus haut-niveau, car les transformations sont guidées par le développeur, ce qui permet d'avoir des implémentations du système bien plus performantes que ce qui est possible avec les langages de très haut-niveau. Cependant, l'intervention du développeur demande du temps et des efforts ce qui augmente la distance cognitive. Malgré tout, avec l'évolution des technologies des compilateurs et des langages, l'intervention du développeur tend à se réduire et la plupart des efforts aujourd'hui sont remplacés par des étapes automatisées.

— **Les architectures logicielles**. Les *architectures logicielles* réutilisables sont des ensembles ou sous-ensembles logiciels à gros-grains qui capturent la structure globale conceptuelle d'un système logiciel. Cette structure globale représente un effort important de conception et d'implémentation qui peut être réutilisé dans son ensemble. La réutilisation à ce niveau offre un levier significatif dans le développement logiciel.

Des exemples d'architectures réutilisables sont par exemple des systèmes de base de données, qui sont adaptés et réutilisés dans différentes applications ; les compilateurs pour lesquels différents analyseurs lexicaux, syntaxiques et générateurs de code peuvent être insérés ; les architectures à tableau noir ou multi-agents ; ainsi que les systèmes de modélisation géométrique ou de visualisation.

A une échelle plus fine, les *design patterns* peuvent être décrits comme des micro-architectures. Gamma et coll. les décrivent comme des descriptions d'objets et de classes communicants qui sont spécialisés pour résoudre des problèmes de conception par rapport à un contexte particulier (Gamma et al., 1995).

Les architectures logicielles sont analogues aux schémas logiciels à gros-grains. Elles se focalisent cependant sur les sous-ensembles et leurs interactions, plutôt que sur les structures de données et les algorithmes. Elles sont aussi proches des générateurs d'applications du fait que les concepts d'organisation de grands sous-systèmes sont réutilisés. Cependant, les générateurs d'application sont souvent des systèmes isolés dont l'architecture est implicite, alors que les architectures logicielles peuvent souvent être explicitement

spécialisées et intégrées à d'autres architectures pour créer beaucoup d'architectures composites différentes.
L'avantage de cet artefact est son haut-niveau d'abstraction que le développeur d'application utilise à partir d'un domaine d'application pour instancier et composer des architectures logicielles. Ainsi, la distance cognitive est faible. De plus, les architectures réutilisables peuvent être utilisées soit indépendamment pour créer des applications utilisateur, ou comme des briques de construction pour créer des architectures logicielles de plus haut niveau. La création d'architectures réutilisables est difficile mais, au vu du succès des design patterns, elle apparaît très utile pour partager des concepts et faciliter la communication entre développeurs.

1.1.2 Les systèmes tableau noir

A partir de la typologie des différentes approches de réutilisation d'artefacts logiciels, proposée par Krueger, nous allons présenter les systèmes tableau noir, qui ont été conçus non seulement pour réutiliser des composants logiciels, mais aussi pour faciliter leurs coopérations et leurs interactions. En particulier, nous décrirons en détail les abstractions utilisées.

Nous avons vu que le domaine de la modélisation des plantes est un domaine pluridisciplinaire faisant appel à des disciplines aussi variées que la modélisation mathématique, l'informatique ou différents domaines de la biologie et de la physique. Développer un modèle de plante structure-fonction nécessite de pouvoir réutiliser des modèles, algorithmes et composants logiciel développés dans ces différentes disciplines. De plus, ces entités logicielles doivent pouvoir interagir entre-elles et s'échanger de l'information pour bénéficier de l'expertise de l'ensemble des entités. Cette problématique de la collaboration entre entités logicielles, proche de celle de la réutilisation logicielle, se retrouve dans plusieurs domaines de recherche. Ces domaines incluent l'intelligence artificielle avec les *systèmes à tableau noir* et les *systèmes multi-agents*, l'ingénierie logicielle avec les *systèmes à composants* et les *systèmes distribués* à petite et grande échelle (e.g. service web, grille de calcul, ...), ainsi que les systèmes de workflows scientifiques et industriels.

Les *système à tableau noir* apparurent dans les années 70 en intelligence artificielle pour essayer d'intégrer des modules logiciels coopérants entre eux (Engelmore et Morgan, 1988 ; Jagannathan, Dodhiawala et Baum, 1989). L'objectif initial était de reproduire le style souple du brainstorming qu'utilise un groupe d'expert, autour d'un tableau noir, pour résoudre un problème qu'aucun d'eux n'aurait pu résoudre seul. Dans ce processus, chaque expert écrit sur le tableau les informations qu'il juge utiles

et pertinentes pour faire avancer la réflexion en vue de trouver une solution. A partir de ces informations, les autres experts aux compétences complémentaires, peuvent à leur tour ajouter des informations en fonction de leurs compétences respectives. Alors qu'aucun des experts n'a les moyens de résoudre le problème, en mettant en commun l'expertise collective, une solution va être trouvée. Cette approche a inspiré la conception des systèmes à entités logicielles collaborantes (SLC) qui impliquent l'intégration et la coordination d'entités logicielles relativement autonomes et indépendantes les unes des autres, et qui sont capables de travailler ensemble. Ce fut le cas des systèmes à tableau noir. Par la suite, la recherche en système multi-agents a revisité ce domaine en mettant l'accent sur une approche centrée agent. Dans cette approche, l'objectif est toujours de pouvoir réaliser une collaboration efficace entre un groupe d'entités logicielles indépendantes, mais sans passer par la structure de données centralisée qu'est le tableau noir.

Un système à tableau noir comporte trois composants principaux (Corkill, 2003) :

— Les **sources de connaissance** (SC) sont des modules de calcul indépendants qui, ensemble, contiennent l'expertise suffisante pour modéliser un système donné ou résoudre un problème. Les SCs peuvent être très différentes entre elles, aussi bien dans leur représentation interne que dans le formalisme de calcul qu'elles utilisent. Elles sont *anonymes*, c'est à dire qu'elles ne communiquent pas directement entre elles et ne connaissent pas les autres SCs qui sont présentes dans le système.

— Le **tableau noir** (TN) est un structure de données globale qui contient les données d'entrée, les résultats de calcul intermédiaires et toutes les données de sortie. L'interaction et la communication entre les SCs se fait par modification du TN.

— Le **contrôleur** choisit, lors de l'exécution, quelles sont les SCs à activer et quelles sont les ressources qu'il doit lui allouer. Le contrôleur est séparé des SC individuelles.

Décrivons en détail ces trois composants principaux :

— **Les sources de connaissance** - Les systèmes à tableau noir utilisent une modularisation fonctionnelle de l'expertise. Chaque SC est un spécialiste pour résoudre certains aspects de l'application entière et est indépendante et séparée des autres SCs. Une SC ne requiert pas la contribution des autres SCs pour pouvoir contribuer. A partir du moment où elle a trouvé, au niveau du TN, l'information dont elle avait besoin, elle peut agir sans l'assistance des autres SCs. Ainsi, sans avoir besoin de changer les SC existantes, une nouvelle SC peut être ajoutée au système et les SCs présentes peuvent être remplacées par d'autres plus efficaces ou peuvent être supprimées. Une SC a une granularité

à gros grain, utilisant les ressources de calcul nécessaires en fonction de sa spécialité. Bien qu'une SC n'ait pas besoin de connaître l'expertise ou la présence d'une autre SC, elle a besoin de connaître la syntaxe et la sémantique des informations qui peuvent lui être utiles et qui sont disponibles sur le TN. Chaque SC connaît les conditions qui lui permettent de contribuer à la résolution du problème et elle essaye de contribuer en fournissant des informations au moment opportun.

— **Le tableau noir** - Le TN est une structure de données partagée qui est accessible à toutes les SCs et sert comme :
  — *un segment de mémoire partagée* contenant i) les données d'entrée, ii) les solutions partielles, les alternatives, les solutions finales, ainsi que iii) les informations utiles au contrôleur.
  — un *moyen de communication et de stockage*.

  Les applications à tableau noir ont tendance à avoir des structures de TN sophistiquées, avec plusieurs niveaux d'abstraction. Bien que cette organisation des données du TN facilite le travail du développeur et de l'utilisateur du système, la raison principale est de pouvoir localiser l'information utile de façon efficace. Si le problème à résoudre est complexe et que les contributions enregistrées au niveau du TN sont nombreuses, il devient difficile d'accéder rapidement à l'information pertinente. Il ne faut pas avoir à parcourir entièrement le TN pour savoir s'il contient les données nécessaires pour pouvoir activer ou non une SC.

  L'accès rapide aux données est donc nécessaire pour pouvoir utiliser le TN comme une mémoire partagée avec les autres SCs ayant contribuées précédemment. Une caractéristique importante des TN est de pouvoir intégrer des contributions non prévues à l'avance. En effet, certaines contributions peuvent se révéler utiles bien plus tard dans le processus de décision, une fois qu'un travail substantiel a été réalisé par d'autres SCs. Ainsi, le système garde la mémoire des contributions et permet d'éviter de les recalculer plus tard, au moment où on en a besoin.

— **Contrôleur** - Dans un système de TN, un mécanisme séparé de contrôle, appelé le contrôleur, coordonne la simulation en permettant à des SCs de répondre aux modifications faites sur le TN.

  En général, bien qu'il existe un très grand nombre de stratégies d'ordonnancement (Liu et al., 2015), les deux stratégies les plus répandues sont le contrôle procédural ou opportuniste. Le contrôle procédural suit une stratégie définie à l'avance, souvent décrite par des instructions impératives dans un langage

procédural. A l'inverse, un contrôle opportuniste suit une stratégie incrémentale où la solution est calculée pas à pas. Ce contrôle est adaptatif. Il évolue au cours de l'exécution.

Un système à TN utilise un contrôleur opportuniste pour pouvoir mener un type de raisonnement incrémental : la solution est calculée pas à pas. Dans les approches classiques de contrôle pour les systèmes de TN, les SCs en cours d'exécution génèrent des événements lorsqu'elles modifient le TN. Ces événements sont conservés et classés jusqu'à ce que l'exécution de la SC soit achevée. A ce moment-là, le contrôleur utilise les événements pour réveiller et activer les SCs. Les contributions potentielles des SCs sont classées. La plus appropriée est sélectionnée pour être exécutée. Ce cycle se termine lorsque le problème est résolu ou la simulation terminée.

Il est important que le contrôleur puisse choisir parmi les SCs sans connaître le détail de chaque SC. Sans une telle séparation, la modularité et l'indépendance des SCs seraient perdues. Si une connaissance particulière de toutes les SCs doit être ajoutée au contrôleur, ce dernier devrait être modifié à chaque fois qu'une SC est ajoutée ou supprimée du système. En même temps, nous ne voulons pas que les SC opèrent des décisions de contrôle de façon autonome. Dans un système à TN, les décisions de contrôle sont déléguées au contrôleur.

1.1.3 Les systèmes de workflows scientifiques

Dans cette partie, nous allons présenter les workflows scientifiques, qui ont choisi une approche différente des systèmes à tableau noir, pour la réutilisation et la coordination des artefacts logiciels. Nous mettrons en particulier l'accent sur l'évolution de ces systèmes au cours de l'histoire récente, les domaines d'application dans lesquels ils sont utilisés ainsi que sur le formalisme et les abstractions qu'ils proposent.

Les systèmes de gestion de workflows scientifiques (Scientific Workflows Management Systems ou SWfMS) ont été introduits pour analyser et simuler des expériences computationnelles complexes, et sont utilisés dans de très nombreux domaines scientifiques, confrontés à un déluge de données sans précédent (Hey, Tansley et Tolle, 2009). On retrouve les prémices de cette approche dans la philosophie Unix, qui s'appuie sur des chaînes de traitement écrites en langage shell, qui permet une grande puissance d'analyse en utilisant un petit nombre de programmes (e.g. grep, sed, awk, ...).

En modélisation des plantes, et particulièrement en analyse de l'architecture des plantes, on retrouve des prémices de l'approche workflow à travers les chaînes de traitements implémentées en langage AML du logiciel AMAPmod. L'objectif du langage AML,

un langage fonctionnel maison, était de créer des scripts d'analyses réutilisables. On retrouvera d'ailleurs, dans la suite de cette thèse, des similarités entre langages fonctionnels et langages de dataflows (Godin et Guédon, 1999).

L'analyse de grandes masses de données impacte particulièrement la biologie à travers la bio-informatique (Cohen-Boulakia et al., 2017), le phénotypage (Pradal et al., 2017 ; Tardieu et al., 2017) ou la bio-médecine (Olabarriaga, Glatard et Boer, 2010). Mais d'autres disciplines comme l'astronomie (Jacob et al., 2009) ou l'éco-informatique (Michener et Jones, 2012) sont aussi concernées. Cependant, les SWfMS, qui sont apparus au début des années 2000, proviennent d'une longue tradition de recherche dans de nombreux domaines de l'informatique.

A la différence des *business workflows* et des *ETL (Extract - Transform - Load) workflows* (Long et al., 2018), les SWfs sont composés de briques de base, les activités (Liu et al., 2015) ou acteurs (Ludäscher et al., 2006), définies par des fonctions complexes fournies par l'utilisateur et sont spécialisés dans la transformation et l'analyse de données (Cohen-Boulakia, 2015 ; Yildiz, Guabtni et Ngu, 2009 ; Cerezo, Montagnat et Blay-Fornarino, 2013). Ils s'appuient sur le formalisme des flux de données (ou dataflow) qui a été introduit au début des années 70. Dans les années 70 et 80, la motivation principale des chercheurs dans ce domaine était d'exploiter le parallélisme massif (Johnston, Hanna et Millar, 2004). A cette époque, l'effort était porté sur la conception d'architecture et de langage de dataflow pour remplacer les machines *Von Neumann* qui souffraient de deux limitations importantes pour le parallélisme massif : i) les variables globales et ii) la mémoire partagée (Backus, 1978). L'alternative proposée fut d'introduire les architectures matérielles de dataflow qui permettaient de répondre à ces limitations en utilisant uniquement de la mémoire locale et en exécutant les instructions dès que les opérandes étaient disponibles (Weng, 1975 ; Davis, 1978). Ces architectures semblaient prometteuses (Dennis, 1980) et un certain nombre de machines ont été réalisées (Davis, 1978). Mais face à l'évolution matérielle, des problèmes sont apparus pour compiler les programmes impératifs sur des architectures de dataflows, en particulier la localité et les effets de bord. En supprimant certains aspects de langages impératifs conventionnels, les chercheurs ont créé de nouveaux langages, les langages de programmation dataflows (Ackerman, 1982), qui s'adaptaient plus facilement aux architectures de dataflows et pouvaient s'exécuter plus efficacement. Les programmes écrits dans ces langages se compilaient en des graphes de dataflow, le langage machine des ordinateurs dataflow.

Cependant ces architectures ne remplacèrent jamais celles plus conventionnelles, du fait entre autre d'un parallélisme à grain trop fin (Veen, 1986). De meilleurs performances pouvaient être obtenues par des approches hybrides VonNeumann/Dataflow

en exploitant un parallélisme à grain plus large en groupant séquentiellement des instructions tout en respectant le formalisme dataflow (Bic, 1990).

Les années 90 virent le développement des langages de programmation visuelle orientés dataflow (Whitley, 1997). Même si ces environnements utilisaient les avantages liés à la parallélisation, leur motivation principale était l'ingénierie logicielle. L'expérience a montré qu'un des intérêts principaux de ces environnements de programmation visuels résidait dans la gestion du cycle de développement logiciel, comprenant des tâches comme la conception, le codage, le débogage et la modification du logiciel (Baroth et Hartsough, 1995 ; Whitley, 1997). Jamal et Wenzel (Jamal et Wenzel, 1995) ont montré, à partir du logiciel propriétaire LabView, l'intérêt des langages de programmation visuels dans des cas réels.

Les langages de dataflows et les environnement de programmation visuels continuent à se développer dans divers domaines applicatifs :

Tout d'abord, en **visualisation scientifique** l'un des environnements pionniers fut l'Application Visualization System ou AVS (Upson et al., 1989). Ce système permettait à des scientifiques de visualiser leurs données 2D ou 3D, en réutilisant des modules et en les connectant à travers l'environnement de programmation visuel. Pour chaque module, une interface graphique permettait de modifier les paramètres. Les modules étaient couplés entre eux en connectant leurs ports d'entrées et de sorties. Les ingénieurs pouvaient quand à eux, étendre les bibliothèques de modules en développant la leur dans le langage C++. Leur bibliothèque étant chargée dynamiquement par l'application. Les concepts d'AVS ont été repris par de nombreux systèmes de visualisation comme Vision (Sanner, Stoffler et Olson, 2002), VisTrails (Callahan et al., 2006), MeVisLab (Koenig et al., 2006) ou encore Voreen (Meyer-Spradow et al., 2009). Tous ces systèmes mettent l'accent sur l'interactivité et sur la possibilité qu'à un utilisateur final de visualiser et d'analyser ses données sans recourir à la programmation (*end-user programming*). D'ailleurs, à la différence d'AVS, ces applications disposent toutes d'un langage interprété, le langage Python, pour étendre le système, plutôt que d'un langage compilé, le C++ dans le cas d'AVS, qui rend plus difficile l'accès à des scientifiques non informaticiens. Dans ce domaine, le formalisme de calcul utilisé par ces applications est le *flux de données* ou *dataflow*. Ce formalisme est particulièrement adapté à la visualisation scientifique qui procède essentiellement par le chargement, la transformation et la visualisation de données. C'est devenu d'ailleur le formalisme de réference depuis que des bibliothèques libres comme VTK l'ont adopté, qui est réutilisée par de très nombreuses applications en visualisation, comme Paraview ou Mayavi.

Dans les domaines de **l'automatique, de l'électronique et du contrôle**, l'application industrielle la plus répandue est le logiciel *LabView* (Wells et Travis, 1997). Cette application fut pionnière dans son domaine et est aujourd'hui encore très utilisée. Au point qu'aujourd'hui, la plupart des industriels proposant des drivers pour des composants électroniques, les fournisse sous la forme d'un dataflow (Virtual Instrument ou VI) pour LabView. LabView est fondé sur le modèle de dataflows. Il transforme de façon statique l'ordonnancement dataflow et produit un code C optimisé. A partir d'une interface de programmation visuelle, il permet aussi d'exporter des vues sous la forme d'une application web qui est utilisée par le scientifique pour contrôler son application.

En **informatique graphique**, les applications utilisées sont des modeleurs permettant de gérer l'ensemble du cycle de vie d'un projet d'infographie, comme par exemple la création d'un film ou d'un jeu vidéo. Les modeleurs les plus connus sont Blender (Hess, 2007) en logiciel libre, et Autodesk Softimage ou Maya dans l'industrie. Ils permettent de générer des scènes 3D, de les texturer, de les animer et d'en produire un rendu réaliste sous la forme d'une image ou d'un film. Ces logiciels sont interactifs. L'utilisateur travaille directement sur la scène en 3D, en construisant par son interaction, une structure de données de graphe de scène, très efficace pour le rendu (Strauss et Carey, 1992). Cependant, les modeleurs construisent un graphe d'activités (workflow) de façon implicite pour permettre de gérer des opérations d'undo/redo, mais aussi permettre à l'utilisateur d'étendre l'application sous forme de macro. Dans ce cas, les activités sont des opérateurs (transformation, texture, création de formes) qui travaillent tous sur une structure de données centrale, le graphe de scène. Ces modeleurs proposent aussi de la programmation visuelle pour permettre à l'utilisateur d'effectuer des tâches très spécialisées comme concevoir de nouveaux shaders pour le rendu.

En **musique**, aussi bien en analyse du signal qu'en composition assistée par ordinateur, la programmation visuelle s'est développée pour permettre aux compositeurs de composer dans un environnement de haut-niveau (Assayag et al., 1999). Les deux applications emblématiques furent le logiciel libre PureData (Puckette, 1996) et OpenMusic, développé en France, à l'IRCAM, dans les années 90 (Assayag et al., 1999). Ces applications utilisent comme modèle le flux de données, avec la possibilité de traiter des signaux continus (ou *data stream*) en entrée. Mais à la différence des applications des autres domaines, elles prennent en compte des structures de données complexes et hétérogènes, et permettent de gérer des dimensions fonctionnelles, spatiales et temporelles. Par exemple, OpenMusic propose à la fois un modèle fonctionnel et temporel à travers trois composants : le *Patch*, la *Maquette* et les *Maquettes*. Le

Patch combine à la fois des objets symboliques (comme des notes et des portées) avec des transformations de données musicales sous la forme de signaux. La *Maquette* est responsable de l'organisation temporelle des différents patchs et les *Maquettes* permettent de construire et d'assembler des hiérarchies pour pouvoir gérer la complexité et la réutilisation de *Patch*.

Finalement, à partir des années 2000, les sciences expérimentales, et en particulier les sciences du vivant, ont dû gérer, analyser, simuler et visualiser d'énormes volumes de données sur des infrastructures de calcul fournissant une puissance inégalée, mais très distribuée (Liu et al., 2015; Görlach et al., 2011). Cette situation a amené les scientifiques à proposer un nouveau terme : la science des données. Or pour pouvoir accéder et bénéficier à ces grandes infrastructures, les informaticiens ont conçu des infrastructures logicielles nommées des e-infrastructures ou cyber-infrastructures. Elles proposent des abstractions de haut-niveau, souvent basées sur le concept de workflow scientifique, qui utilisent le modèle de flux de données. Parmi les workflows scientifiques les plus utilisés on trouve entre autre Galaxy (Goecks, Nekrutenko et Taylor, 2010), Taverna (Oinn et al., 2004), Kepler (Ludäscher et al., 2006), Pegasus (Deelman et al., 2005), Swift (Wilde et al., 2011) ou encore Nextflow (Di Tommaso et al., 2017).

D'un point de vue plus conceptuel, un workflow scientifique est l'assemblage d'un ensemble d'activités de calcul liées entre elles par des dépendances représentant le flux de données (Deelman et al., 2009). Un workflow peut avoir une structure hiérarchique. Une activité peut être soit une activité atomique (e.g. un programme, un service web ou une fonction), soit contenir un sous-workflow. Un sous-workflow est donc à la fois un workflow et une activité. Les workflows peuvent être représentés de différentes façons. Cependant, la représentation la plus générale est celle d'un *graphe dirigé*, dont les sommets représentent les activités avec des ports d'entrée et de sortie et les arrêtes le flux de données dirigé entre le port de sortie de l'activité source et d'entrée de l'activité cible. Peu de systèmes peuvent gérer des cycles dans les workflows. Les graphes sont donc la plupart du temps acycliques (Directed Acyclic Graph ou DAG), à l'exception des systèmes permettant de représenter d'autres modes de calcul que les dataflows, comme dans Kepler avec le modèle de calcul *machine à état fini*.

Un *modèle de calcul* (Model Of Computation ou MOC) est un modèle permettant de coordonner ou de chorégraphier l'exécution des activités d'un workflow. La plupart des systèmes de workflows n'utilisent qu'un modèle de calcul, en général de type dataflow. Cependant, Kepler, tout comme PtolemyII avant lui, propose un ensemble de MOCs. Dans ces systèmes, les activités sont invoquées par un chorégraphe ou

chef d'orchestre (une forme d'ordonnanceur) qui est associé à un workflow ou sous-workflow et implémente un MOC (Lee, Neuendorffer et Wirthlin, 2003).

Un grand nombre de modèles de calcul ont été décrits par l'équipe d'Edward Lee (Lee, Neuendorffer et Wirthlin, 2003 ; Lee, 2010). Son idée a été d'utiliser une syntaxe abstraite commune à tous ces MOCs et de les composer de façon hierarchique en utilisant une sémantique abstraite. Il propose de modéliser des systèmes complexes par composition de modèles hétérogènes, en utilisant des workflows hiérarchiques d'acteurs (ou d'activités). Si une activité contient un sous-workflow avec le même modèle de calcul que le workflow parent, elle est dite transparente. Le groupement d'acteurs en sous-workflow ne sert qu'à simplifier la complexité et favoriser la réutilisation. Par contre, si son modèle de calcul est différent, il est considéré comme opaque. Il est vu par le workflow parent le contenant comme une boîte noire. C'est ce concept d'opacité qui est le facteur déterminant pour gérer l'hétérogénéité hiérarchique. Ainsi des modèles ayant différentes sémantiques (temps continu, temps discret, dataflow) peuvent être composés au sein d'un même workflow. On retrouve ce même concept dans le formalisme DEVS avec le principe de fermeture par composition (Zeigler, Praehofer et Kim, 2000).

Dans ce formalisme, les activités s'exécutent de façon concurrentes, recevant des données des autres activités en leurs ports d'entrée et en renvoyant des données aux autres acteurs à travers leurs ports de sortie. C'est le modèle de calcul qui définit ce que signifie exactement *s'exécuter de façon concurrente*, ainsi que la façon dont les données sont envoyées et reçues. Ce modèle a une sémantique concrète. On parle de la sémantique d'un modèle de calcul pour insister sur le fait que, lorsque différents MOC seront associés au même graphe d'acteurs, les résultats seront différents. L'exécution du graphe sera différente car à chaque MOC peut-être associé un contrôle de l'exécution, de la communication et un modèle de temps particulier.

Pour composer de façon hiérarchique des modèles de calculs, on a besoin d'associer à chacun de ces modèles une sémantique abstraite. Elle est constituée de trois aspects : le contrôle de l'exécution, la communication et le modèle de temps. Pour plus de détails, un modèle formel de la sémantique abstraite a été décrit par Lee et Sangiovanni-Vincentelli (Lee et Sangiovanni-Vincentelli, 1998).

Finalement, les workflows scientifiques permettent de gérer l'automatisation, le passage à l'échelle, la réutilisation et la provenance (soit ASAP en anglais pour Automation, Scalable, Adaptation and Provenance) (Cuevas-Vicenttín et al., 2012) :

— **Automatisation** - Un workflow capture toutes les étapes d'une analyse ou d'une simulation. Il permet de formaliser une expérience computationnelle

en explicitant chacune des tâches et en recalculant *automatiquement* son exécution lors d'une modification d'étapes (activités) ou de paramètres.

— **Passage à l'échelle** - Les workflows doivent pouvoir passer à l'échelle et être tolérants aux erreurs pour pouvoir gérer de grands jeux de données et pour aborder des expériences computationnelles demandant des calculs intensifs. Ils permettent de gérer de très grands volumes de données et d'utiliser les très grandes infrastructures de calcul distribué comme les clusters, les grilles de calcul et le cloud. Cela est permis parce qu'ils offrent une abstraction qui permet de découpler la déclaration des tâches ou activités à exécuter de leurs exécutions, grâce à divers modèles de calculs.

— **Abstraction, Evolution, Réutilisation** - Les environnement de programmation visuelle et les applications web comme celle offerte par *Galaxy* permettent de simplifier l'utilisation des workflows par des non-scientifiques. De plus, la réutilisation est aussi facilitée par les formalismes hiérarchiques et par ces environnements. Réutiliser une tâche consiste la plupart du temps à une action de glisser / déposer d'une activité représentant un programme ou un workflow. Des entrepôts de workflows, comme myExperiment (Goble et al., 2010), permettent de partager des workflows entre plusieurs utilisateurs et de les analyser. Des annotations peuvent aussi faciliter la réutilisation.

— **Provenance** - De nombreux systèmes permettent d'enregistrer la provenance (historique de l'exécution et provenance de la donnée) (Davidson et al., 2007 ; Cohen-Boulakia et al., 2017). Cela permet de répondre à la question : quelles sont les étapes, les algorithmes, les paramètres et les données qui m'ont permis d'obtenir ce résultat ?

1.2 Les limites actuelles

Nous avons présenté trois approches qui abordent le problème de la modularité et de la réutilisation. Krueger a formalisé, à travers une taxonomie de méthodes, les différentes formes de réutilisation logicielle. Sa contribution majeure a été de montrer l'importance de choisir la bonne abstraction pour favoriser la réutilisation. Lorsque l'abstraction est trop éloignée du domaine considéré, l'expression des concepts du domaine dans cette abstraction est rendue plus difficile. Par contre, si l'on choisit une abstraction difficile à convertir en code operationnel, les mécanismes de transformation de cette abstraction en code réutilisable risquent d'être difficiles à mettre en oeuvre. Ils vont donc limiter son utilisation du fait de son manque d'efficacité. Ainsi, un langage de très-haut niveau va être facilement adopté par des scientifiques non informaticiens, mais va être difficile à transformer en code exécutable efficace.

L'enjeu est donc de trouver les bonnes abstractions qui seront à la fois expressives pour un domaine donné, mais aussi opérationnelles et faciles à mettre en oeuvre.

En modélisation des plantes, une des abstractions qui fut un grand succès sont les L-systèmes. En proposant un langage déclaratif, couplé à un langage généraliste (L+C), Pr. Prusinkiewicz et ses collègues ont permis d'exprimer le développement d'une plante de façon concise en 3D. Ainsi, en utilisant des concepts proches de la botanique, un modélisateur peut exprimer, à partir de règles locales et dans un langage de réécriture, la croissance et le développement d'une plante à partir de la production des apex. Son formalisme est fondé sur la réécriture de chaînes de caractères, représentant des arborescences.

Malgré la puissance d'expression de ce formalisme et du fait de l'évolution des disciplines et des questions scientifiques, les besoins en modélisation ont évolués. Le problème s'est déplacé de la modélisation d'une architecture en croissance, à un problème de biologie intégrative, nécessitant d'intégrer au sein d'un même modèle les contributions de plusieurs équipes pluri-disciplinaires et pouvant travailler à différentes échelles de représentation. Les L-systèmes ne permettaient pas facilement d'intégrer des processus développés dans d'autres formalismes, qu'ils soient externes, comme par exemple les modèles de transfert radiatif ou de dispersion de maladies, ou internes, comme des flux d'eau ou d'auxine. Pour ce faire, le formalisme a évolué afin de pouvoir aborder ces cas (Prusinkiewicz et al., 2009 ; Cieslak, Seleznyova et Hanan, 2011). Mais, dans un même temps, d'autres systèmes ont vu le jour.

L'équipe du Pr. Kurth a étendu le formalisme des L-systèmes en développant le langage XL, une extension du langage orienté-objet (Java), pour pouvoir faire de la réécriture de graphe, plutôt que d'aborescence, dans un cadre plus général (Kniemeyer, Buck-Sorlin et W., 2004 ; Hemmerling et al., 2008). Tout comme les L-systèmes, cette approche a été implémentée dans une plateforme de modélisation logicielle libre, GroIMP (Kniemeyer et Kurth, 2008). Ces dernières années, ce formalisme a été étendu pour pouvoir modéliser des plantes à différentes échelles (Ong et al., 2014) ou résoudre des équations différentielles de façon plus intuitive (Hemmerling et al., 2013). Malgré le succès de ces différents formalismes et plateformes de modélisations auprès des modélisateurs, le partage, la diffusion et l'intégration de modèles existants développés en-dehors de ces formalismes restent très difficiles pour plusieurs raisons :

1. La structure centrale de ces plateformes (arborescence ou graphe) est implicite et ne peut être partagée par des processus externes. Cela empêche le développement de systèmes comme les tableaux noirs, par exemple.

2. Ces environnements ne fournissent aucune abstraction pour ordonnancer des composants de façon explicite. Le seul ordonnanceur disponible est le formalisme proposé (i.e. L-systèmes ou réécriture de graphe). Cela ne permet pas de faire de la multi-simulation, c'est à dire de coupler des composants utilisants différents formalismes.
3. Les mécanismes d'extension sont spécifiques à chaque environnement. Bien qu'un système de plug'in soit disponible dans GroIMP, par exemple, il faut développer les composants en Java et les ajouter à la plateforme de façon intrusive. Sachant que la notion même de composant n'est explicite dans aucune de ces plateformes.

Pour conclure, bien que des formalismes de modélisation aient un grand succès en modélisation des plantes et soient le paradigme dominant, des limites restent présentes pour pouvoir assembler des modèles intégratifs à partir de sources de connaissances hétérogènes. Des solutions conceptuelles ont été proposées dans la littérature et ont été expérimentées dans différents domaines, mais elles ne répondent pas directement à notre problématique. L'étude des travaux de Krueger suggère qu'il faille trouver les bonnes abstractions pour modéliser le développement et la croissance des plantes. Les Workflows permettent d'assembler des sources de connaissances hétérogènes pour faire de l'analyse de données. Mais ils ne permettent pas de représenter la rétro-action entre structure et fonction et ne sont pas utilisés en simulation. Finalement, les systèmes à tableau noir permettent d'orchestrer et de faire coopérer des sources de connaissances hétérogènes, mais souvent le contrôle est opportuniste, bien que de nombreux travaux ont portés sur différentes stratégies de contrôle des tableaux noirs.

1.3 Objectif et Plan de la thèse

L'objectif de cette thèse est de proposer et de mettre en oeuvre, en s'appuyant sur ce constat, un ensemble d'abstractions pour permettre la modularité en modélisation structure-fonction des plantes. Nous proposons un système, étendant le système de tableau noir à contrôle procédural, basé sur une architecture de dataflow.

Tout d'abord, nous présenterons OpenAlea, un système de workflow pour la modélisation des plantes à plusieurs échelles, introduisant pour la première fois l'utilisation d'un modèle de dataflow en modélisation des plantes et l'assemblage de composants hétérogènes (Chapitre 1).

Dans le chapitre 2, nous présenterons PlantGL, une librairie informatique pour la modélisation 3D des plantes à différentes échelles. Cela nous permettra d'introduire la

notion de structure centrale, ou tableau noir, spatialisée, en choisissant une variété topologique définie à différentes échelles, le MTG (ou Multiscale Tree Graph). La géométrie sera obtenue par un plongement de cette structure topologique, ce qui permet de bien séparer information topologique et géométrie, et ce à différentes échelles.

Alors que le chapitre 1 présente la notion de composants comme source de connaissance et que le chapitre 2 introduit la notion de TN multi-échelle, prenant en compte de façon distincte sa structure (topologie) et sa spatialisation (géométrie), le chapitre 3 présente un nouveau modèle d'orchestration, ou contrôleur, inspiré du λ-calcul, permettant à la fois de coupler analyse et simulation, tout en préservant la modularité.

Finalement, les chapitre 4 et 5 seront deux chapitres applicatifs appliquant les concepts développés dans les chapitres précedents en modélisation des plantes. Le chapitre 4 présentera un modèle géométrique et fonctionnel de développement d'une feuille de graminée et son intégration en tant que composant au sein de modèles structure-fonction. Le chapitre 5 présentera l'utilisation du formalisme développé dans cette thèse pour modéliser, de façon générique et modulaire, un système plante / maladie foliaire en épidémiologie. Dans ce système, chaque source de connaissances (la plante, les processus micro-climatiques, les maladies foliaires) est représentée sous la forme d'un composant. Chaque composant est autonome et ne communique qu'indirectement, à travers la structure centrale, ou tableau noir. Finalement, à la différence des systèmes à tableau noir, l'ordonnancement des composants n'est pas opportuniste, mais est procédural et défini explicitement par le modélisateur à l'aide d'un workflow scientifique et de son modèle de calcul associé.

Chapitre 2

OpenAlea : une plateforme logicielle pour la modélisation des plantes

Ce chapitre présente la plateforme **OpenAlea**, une plateforme libre de modélisation structure-fonction des plantes à différentes échelles, du tissu à la plante entière.

La modélisation des plantes est un domaine en pleine évolution, aussi bien dans le domaine de l'agronomie pour comprendre l'impact du changement climatique, qu'en biologie du développement pour comprendre l'impact des interactions entre la forme de la plante résultante de son génotype et de l'environnement, qui interagit avec les fonctions biologiques et modifie en retour le développement de la plante.

La plante peut-être modélisée comme un système complexe où interagissent des composants à différentes échelles de représentation. Comprendre et caractériser les interactions entre les processus de développement et les fonctions biologiques nécessite l'intégration de connaissances disponibles dans différentes disciplines, aussi bien en biologie, en informatique, en mathématique qu'en physique. De plus, l'apparition de nouveaux capteurs en imagerie et l'émergence des plateformes robotisées de génotypage et de phénotypage de plantes (en chambre de culture, en serre ou au champ) ont permis d'acquérir de très grands volumes de données à très haut-débit. Pour traiter ce déluge de données, le recours à la modélisation informatique devient une nécessité.

Cependant, un grand nombre de formalismes sont utilisés pour modéliser les plantes, en fonction de l'échelle d'intérêt (tissu, organe, plante entière ou peuplement), des processus modélisés et de la prise en compte de la structure de façon implicite ou explicite.

Le développement de modèles de plante requiert un recours croissant à la modélisation informatique. Ces modèles sont développés par des équipes internationales, travaillant chacune avec des objectifs et dans des contextes différents. Des infrastructures logicielles efficaces et flexibles sont nécessaires pour améliorer l'interaction

entre ces modèles, leur réutilisation et leur diffusion, ainsi que permettre de les comparer entre eux sur les mêmes bases de données.

Pour cela, nous avons conçu la plateforme OpenAlea, une plateforme modulaire à composants logiciels réutilisables, dont la composition et l'assemblage se fait à l'aide du formalisme des workflows scientifiques basés sur un modèle de calcul *dataflow*. L'architecture du système est construite autour d'un langage interprété de haut-niveau, le langage Python. Ce langage, orienté-objet et généraliste, est largement utilisé dans differents domaines scientifiques et facilite donc la réutilisation de bibliothèques et d'algorithmes développés dans d'autres disciplines. Cela nous permet aussi de diffuser plus largement nos propres méthodes.

Ma contribution principale dans ce travail a été :

- L'introduction du langage Python, comme langage de modélisation et d'intégration de composants hétérogènes, dans la communauté de modélisation des plantes.
- Le développement d'une plateforme modulaire à base de composants réutilisables.
- L'animation d'une communauté libre de recherche pour la modélisation des plantes.

L'impact de la plateforme OpenAlea peut être mesuré par le nombre de citations (264 source google.scholar) et par un nombre important de collaborations, en France et à l'étranger. Le nombre de téléchargement de la plateforme et de ses composants dépasse les 750.000 téléchargements (source gforge.inria.fr), en faisant le logiciel le plus téléchargé sur le site gforge de l'Inria (source gforge.inria.fr).

Ce chapitre est la version originale du papier :

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

C. Pradal, S. Dufour-Kowalski, F. Boudon, C. Fournier, C. Godin.
Publié dans le journal Functional Plant Bioology, 35, 2008, pp 751-760

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

 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.

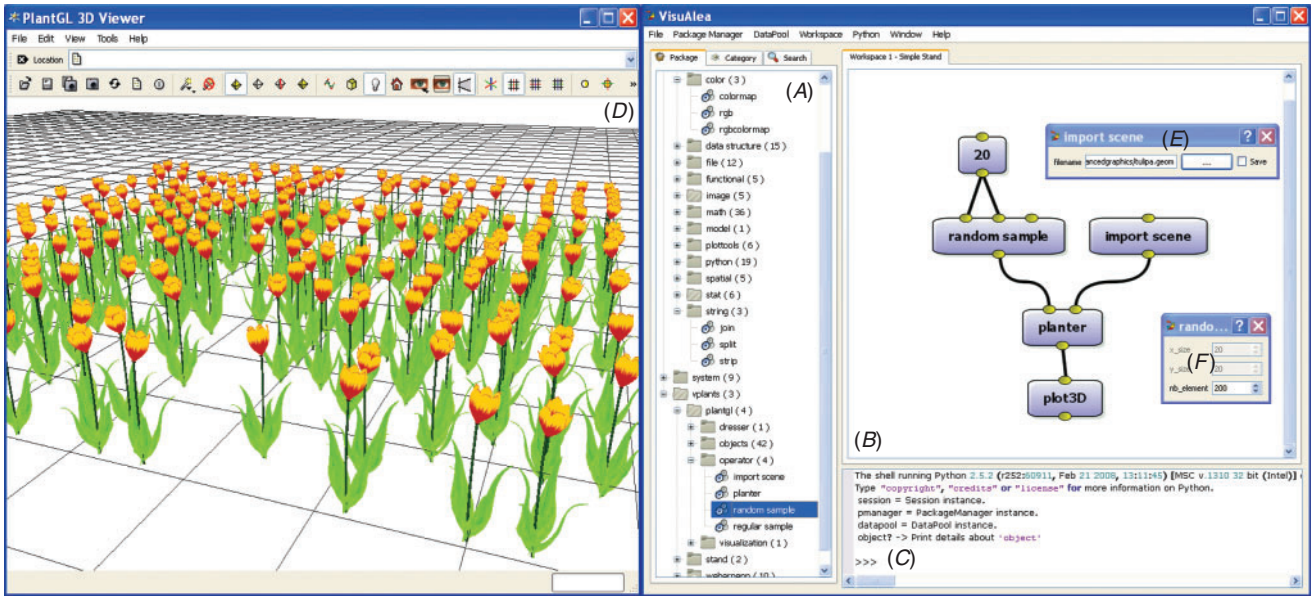

Fig. 1. Snapshot of the OpenAlea visual modelling environment. (*A*) The package manager list packages and nodes found on the system. (*B*) The graphical programming interface enables users to build visual dataflow by interconnecting nodes. A 3-D scene is built by associating a single geometry with a 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

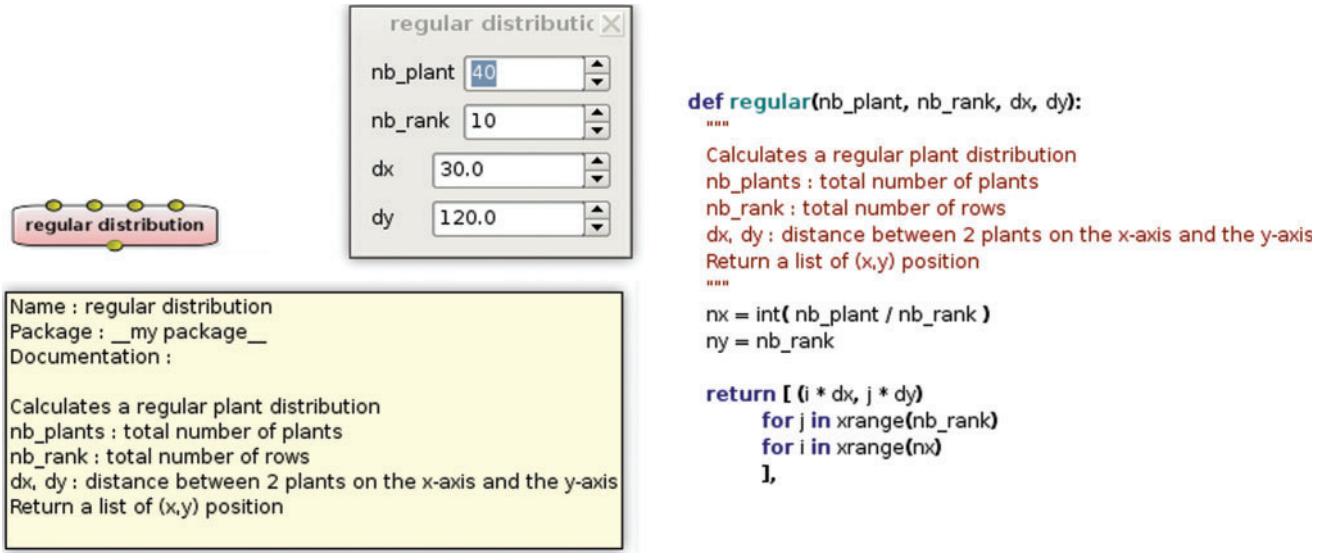

Fig. 2. A graphical node is a visual representation of a function. Input ports at the top represent the input arguments and output ports at the bottom, the resulting values. In this example, the 'regular' node generates a list of position (x, y) corresponding to a regular plant distribution. Documentation 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.

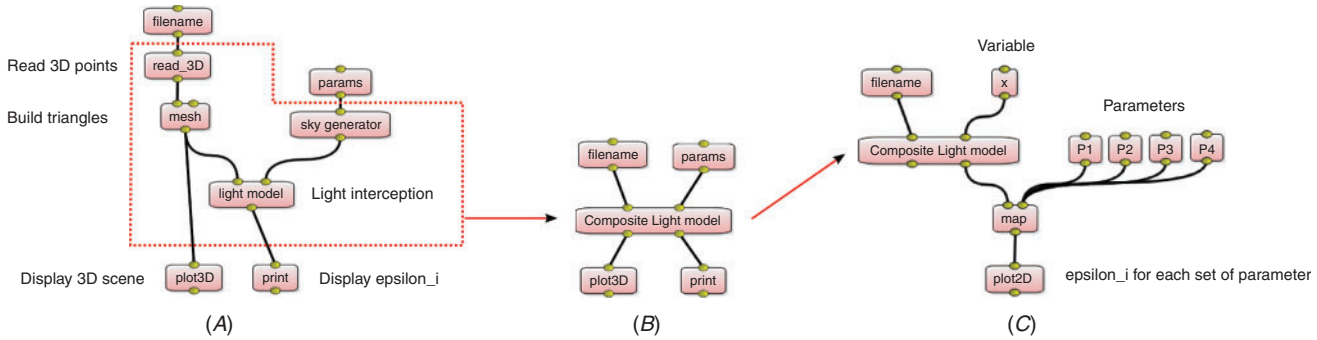

Fig. 3. (*A*) In the first example, we construct a plant model from a set of 3-D points read in a file. (*B*) Then, the light interception is computed using a sky description. The 3-D plant is displayed in a 3-D viewer, and the results of the light model are displayed in the shell. In the second example, 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

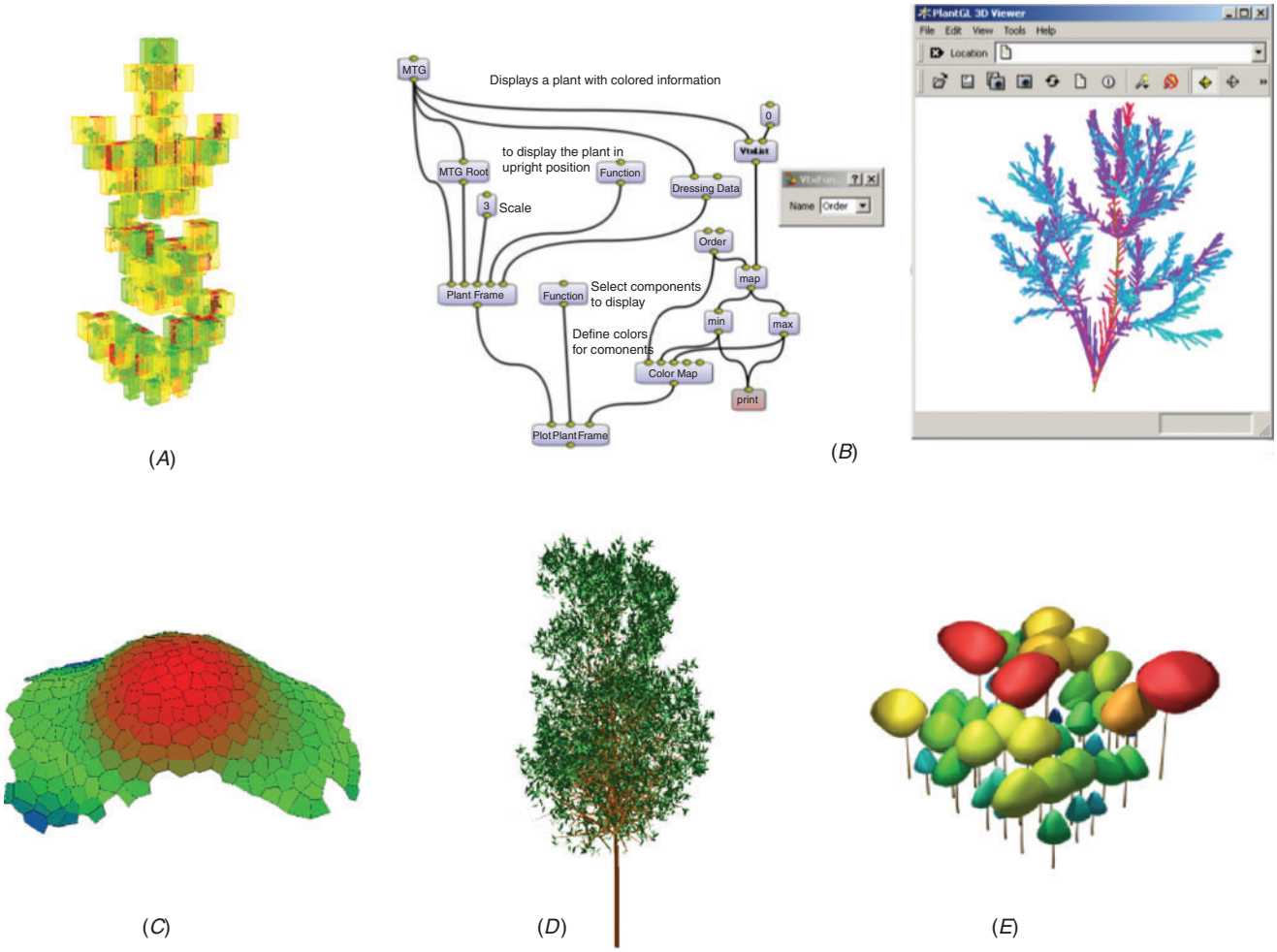

Fig. 4. Example of components integrated in OpenAlea. (*A*) Estimation of the fractal dimension of a plant foliage using the box counting method (Da Silva *et al*. 2006) (*B*) A visual programming example used to explore the topology and geometry of multiscale plant databases using VPlants components. (*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

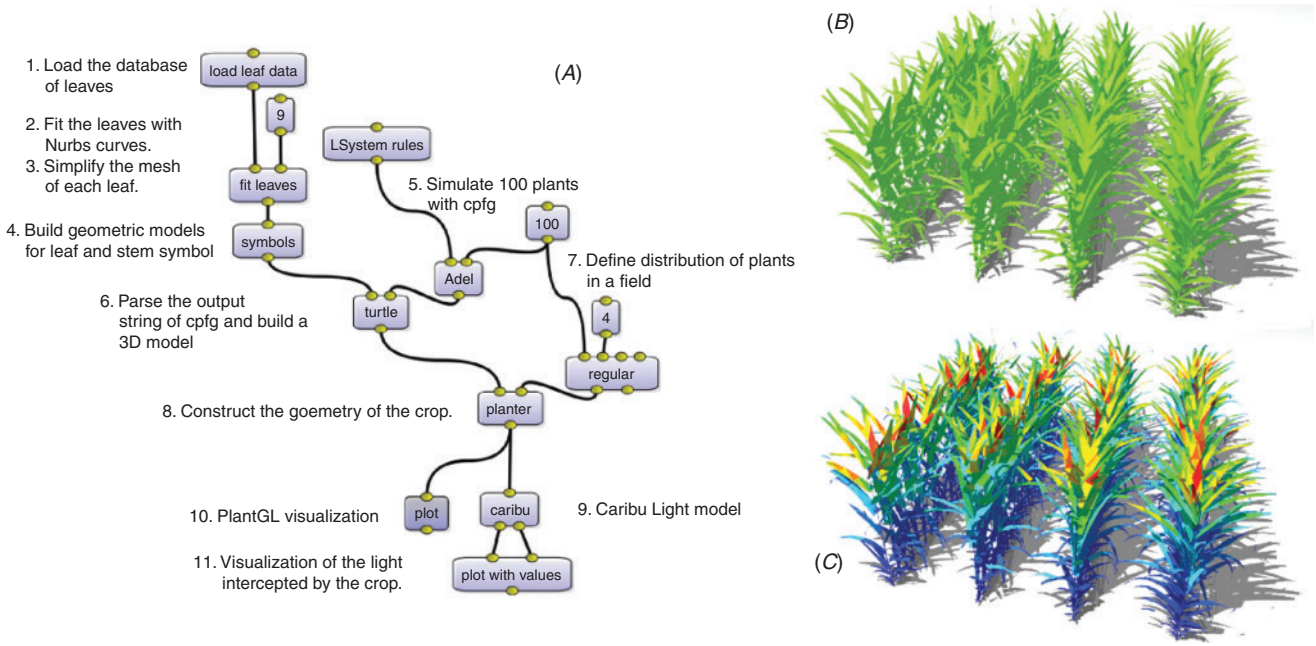

Fig. 5. Snapshots of (*A*) the VisuAlea dataflow, and (*B*) two outputs of an application allowing to reconstruct a maize canopy, and (*C*) to estimate light distribution within it. Annotations on the dataflow succinctly describe the functions of the different nodes. Nodes 1–4 defines the leaf shape 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

Chapitre 3

PlantGL : Modélisation géométrique des plantes multi-échelles

Ce chapitre présente **PlantGL**, une bibliothèque géométrique utilisée pour la création, la simulation et l'analyse de modèles géométriques de plantes 3D à différentes échelles.

PlantGL permet de construire et de manipuler interactivement des modèles géométriques de plantes ou d'organes depuis l'échelle tissulaire jusqu'à l'échelle du peuplement.

Cette bibliothèque libre, qui s'appuie sur une structure de données de graphe de scène, permet de combiner des primitives généralistes et adaptées aux plantes comme des cylindres généralisés ou des enveloppes variées. Elle fournit de nombreux algorithmes et, bien qu'implémentée en C++, propose une interface Python qui permet une grande souplesse d'utilisation.

En plus de la description de l'architecture de cette bibliothèque 3D, nous présentons un ensemble d'exemples tels que la reconstruction d'enveloppes foliaires de grands arbres, le calcul de la dimension fractale par la méthode du comptage des boîtes, l'analyse géométrique du développement de la couronne d'un eucalyptus au cours de sa croissance, ainsi que la simulation de plantes à l'échelle de l'organe, d'un arbre isolé et d'un peuplement.

D'un point de vue théorique, l'apport de ce chapitre est de proposer un couplage entre topologie multi-échelles et géométrie, tout en séparant explicitement la représentation de la topologie de la géométrie. Le MTG (ou Multiscale Tree Graph) permet de représenter des systèmes ramifiés à plusieurs échelles. Ce formalisme, initialement proposé par Godin et Caraglio en 1998 (Godin et Caraglio, 1998) et implémenté dans le logiciel AMAPmod, a été étendu pour devenir la structure centrale d'OpenAlea, aussi bien pour acquérir, que pour simuler des systèmes ramifiés. Dans ce chapitre,

nous étendons le concept de graphe de scène pour les plantes sous la forme d'un *p-scene-graph*. Ce *p-scene-graph* multi-échelles permet d'étendre le formalisme du MTG en ajoutant des informations géométriques. Les différentes échelles d'organisation offre des vues différentes à diverses résolutions du même objet plante. De plus, cette organisation topologique à différentes échelles permet de définir des contraintes au sein d'une même échelle et entre les échelles.

Ma contribtion dans ce travail a été de :

- Formaliser mathématiquement divers opérateurs de génération d'enveloppes géométriques.
- Généraliser l'interprétation des contraintes inter et intra échelles pour reconstruire la géométrie d'une plante mesurée (e.g. algorithme PlantFrame).
- Favoriser la diffusion de cette bibliothèque par son utilisation simplifiée à travers le langage Python.

L'impact de cette publication et du logiciel PlantGL est reconnu par un grand nombre de citations (100 citations source google scholar), aussi bien à l'échelle du méristème que de la plante entière ou des applications en phénotypage. C'est un composant principal de la plateforme OpenAlea, qui est réutilisé par un grand nombre de modèles.

Ce chapitre est la version originale du papier :

PlantGL : a Python-based geometric library for 3D plant modelling at different scales

C. Pradal, F. Boudon, C. Nouguier, J. Chopard, C. Godin. Publié dans le journal Graphical Models, 71(1), 2009, pp 1-21

Graphical Models 71 (2009) 1–21

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

PlantGL: A Python-based geometric library for 3D plant modelling at different scales

C. Pradal [a,1], F. Boudon [a,*,1], C. Nouguier [a], J. Chopard [b], C. Godin [b]

[a] CIRAD, Virtual Plants INRIA Project-Team, UMR DAP, TA A-96/02, Avenue Agropolis, F-34398 Montpellier, France
[b] INRIA, Virtual Plants INRIA Project-Team, UMR DAP, F-34398 Montpellier, France

ARTICLE INFO

Article history:
Received 13 August 2007
Received in revised form 6 October 2008
Accepted 9 October 2008
Available online 26 October 2008

Keywords:
Graphic library
Virtual plants
Crown envelopes
Plant architecture
Canopy reconstruction
Plant scene-graphs

ABSTRACT

In this paper, we present PlantGL, an open-source graphic toolkit for the creation, simulation and analysis of 3D virtual plants. This C++ geometric library is embedded in the Python language which makes it a powerful user-interactive platform for plant modeling in various biological application domains.

PlantGL makes it possible to build and manipulate geometric models of plants or plant parts, ranging from tissues and organs to plant populations. Based on a scene graph augmented with primitives dedicated to plant representation, several methods are provided to create plant architectures from either field measurements or procedural algorithms. Because they are particularly useful in plant design and analysis, special attention has been paid to the definition and use of branching system envelopes. Several examples from different modelling applications illustrate how PlantGL can be used to construct, analyse or manipulate geometric models at different scales ranging from tissues to plant communities.

1. Introduction

The representation of plant forms in computer scenes has long been recognized as a difficult problem in computer graphics applications. In the last two decades, several algorithms and software platforms have been proposed to solve this problem with a continuously improving level of efficiency, e.g. [1–9]. Due to the increasing use of computer models in biological research, the design of 3D geometric models of plants has also become an important aspect of various biological applications in plant science, e.g. [10–16]. These applications raise specific problems that derive from the need to represent plants with a botanical or geometric accuracy at different scales, from tissues to plant communities. However, in comparison with computer graphics applications, less effort has been devoted to the development of geometric modelling systems adapted to the requirements of biological applications.

In this context, the most successful and widespread plant modelling system has been developed by P. Prusinkiewicz and his team since the late 80's at the interface between biology and computer graphics. They designed a computer platform, known as L-Studio/VLab, for the simulation of plant growth based on L-systems [17,1,3]. This system makes it possible to model the development of plants with efficiency and flexibility as a process of bracketed-string rewriting. In a recent version of LStudio/VLab, Karwowski and Prusinkiewicz 18 changed the original cpfg language for a compiled language, L+C, built on the top of the C++ programming language. The resulting gain of expressiveness facilitates the specification of complex plant models in L+C [19]. An alternative implementation of a L-system-based software for plant modeling was designed by W. Kurth [20] in the context of forestry applications. This simulation system, called GroGra, was also recently re-engineered in order to model the development of objects more complex than bracketed strings. The

* Corresponding author.
E-mail address: frederic.boudon@cirad.fr (F. Boudon).
[1] These authors contributed equally to this work.

1524-0703/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2008.10.001

resulting simulation system, GroIMP, is an open-source software that extends the chain rewriting principle of L-Systems to general graph rewriting with relational graph growth (RGG), [21,22]. Similarly to L+C, this system has been defined on top of a widely used programming language (here Java). Non-language oriented platforms were also developed. One of the first ones was designed by the AMAP group. The AMAP software [2,23] makes it possible to build plants by tuning the parameters of a predefined, hard-coded, model. Geometric symbols for flowers, leaves, fruits, *etc.*, are defined in a symbol library and can be modified or created with specific editors developed by AMAP. In this framework, a wide set of parameter-files has been designed corresponding to various plant species. In the context of applications more oriented toward computer graphics, the XFrog software [5,24] is a popular example of a plant simulation system dedicated to the intuitive design of plant geometric models. In XFrog, components representing basic plant organs like leaves, spines, flowerlets or branches can be multiplied in space using high-level multiplier components. Plants are thus defined as graphs representing series of multiplication operations. The XFrog system provides an easy to use, intuitive system to design plant models, with little biological expertise needed.

Therefore, if accuracy, conciseness and transparency of the modeling process is required, object-oriented, rule-based platforms, such as L-studio/VLab or GroIMP, are good candidates for modelers. If interactive and intuitive model design is required, with little biological expertise, then component-based systems, like XFrog, or sketch-based systems are the best candidates. However, if easiness to explore and mathematically analyse plant scenes is required, none of the above approaches is completely satisfactory. Such an operation requires high-level user interaction with plant scenes and dedicated high-level mathematical primitives. With this aim, our team developed the AMAPmod software [25] several years ago, and its most recent version, VPlants, which enables modelers to create, explore and analyse plant architecture databases using a powerful script language. In a way complementary to L-Studio/VLab, VPlants allows the user to efficiently analyse plant architecture databases and scenes from many exploratory perspectives in a language-based, interactive, manner [26–29]. The PlantGL library was developed to support geometric processing of plant scenes in VPlants, for applications ranging from computer graphics [30,31] to different areas of biological modeling [32–34,15,35,36]. A number of high-level requirements were imposed by this context. Similarly to AMAPmod/VPlants, the library should be open-source, it should be fully compatible with the data structure used in AMAPmod/VPlants to represent plants, *i.e.* multi-scale tree graphs (MTGs), it should be accessible through an efficient script language to favor interactive exploration of plant databases, it should be easy to use for biologists or modellers and should not impose a particular modelling approach, it should be easily extended by users to progressively adapt to the great variety of plant modelling applications, and finally, it should be interoperable with other main plant modelling platforms.

These main requirements lead us to integrate a number of new and original features in PlantGL that makes it particularly adapted to plant modelling. It is based on the script language Python, which enables the user to manipulate geometric models interactively and incrementally, without compiling the scene code or recomputing the entire scene after each scene modification. The embedding in Python is critical for a number of additional reasons: (i) the modeller has access to a powerful object-oriented language for the design of geometric scenes, (ii) the language is independent of any underlying modelling paradigm and allows the definition of new procedural models, (iii) high-level manipulations of plant scenes enable users to concentrate on application issues rather than on technical details, and (iv) the large set of available Python scientific packages can be freely and easily accessed by modelers in their applications. From a contents perspective, PlantGL provides a set of geometric primitives for plant modelling that can be combined in scene-graphs dedicated to multiscale plant representation. New primitives were developed to address biological questions at either macroscopic or microscopic scales. At plant scale, envelope-based primitives have been designed to model plant crowns as volumetric objects. At cell scale, tissue objects representing arrangements of plant cells enable users to model the development of plant tissues such as meristems. Particular attention has been paid to the design of the library to achieve a high-level of reuse and extensibility (e.g. data structures, algorithms and GUIs are clearly separated). To favor the exchange of models and databases between users, PlantGL can communicate with the other modelling platforms such as LStudio/VLab and is available under an open-source license.

In this paper, we present the PlantGL geometric library and its application to plant modelling. Section 2 describes the design principles and rationales that underly the library architecture. It also briefly introduces the main scene graph structure and the different library objects: geometric models, transformations, algorithms and visualization components. Then, a detailed description of the geometric models and methods dedicated to the construction of plant scenes is provided in Section 3. This includes the modeling of organs, crowns, foliage, branching systems and plant tissues. A final section illustrates how PlantGL components can be used and assembled to answer particular questions from computer graphics or biological applications at different levels of a modelling approach: creating, analysing, simulating and assessing plant models.

2. PlantGL design and implementation

A number of high-level goals have guided the design and development of PlantGL to optimize its reusability and diffusion:

- *Usefulness*: PlantGL is primarily dedicated to researchers in the plant modelling community who do not necessarily have any *a priori* knowledge in computer graphics. Its interface with modellers and end-users should be intuitive with a short learning curve.

- *Genericity*: PlantGL should not impose a particular modelling paradigm on users. Rather, it should allow them to design their own approach in a powerful way.
- *Quality*: Quality is a major aspect of software diffusion and reusability. PlantGL should therefore be developed with software quality guarantees.
- *Interoperability*: PlantGL should also be interoperable with various plant modelling systems (e.g. L-studio/VLab, AMAP, *etc.*) and graphic toolkits (e.g. Pov-Ray, Vrml, Blender, *etc.*).
- *Portability*: PlantGL should be available on major operating systems (e.g. Linux, Microsoft Windows, MacOSX).

In this section we detail how these requirements have been translated into choices at different levels of the system design.

2.1. Software design

The system architecture derives from a set of key design choices:

- *Open-source*: PlantGL is an open-source software that may be freely used and extended.
- *Script-language based system*: PlantGL is built on the top of the powerful script language, Python. The use of a script language allows users to have a high level of computational interaction with their models.
- *Software engineering*: Object-oriented design is useful to organize large computational projects and enhance code reuse. However, designing reusable and flexible software remains a difficult task. We addressed this problem by using advanced software engineering tools such as *design patterns* [37].
- *Modularity*: PlantGL is composed of several independent modules like a geometric library, GUI components and Python wrappers. They can be used alone or combined within a specific application.
- *Hybrid system*: Core computational components of PlantGL are implemented in the C++ compiled language for performance. However, for flexibility of use, these components are also exported in the Python language.

Among all the available script languages, Python was found to present a unique compromise between the following features: it is (a) open-source; (b) available on the main operating systems; (c) object-oriented; (d) simple to use with syntax sufficiently intuitive for non-computer scientists (e.g. for biologists); (e) interactive: it allows direct testing of pieces of code without requiring a compilation process; and (f) has excellent support for integrating code written in compiled languages (e.g. C, C++, Fortran). Additionally, the Python community is large and very active and a large number of scientific libraries are available and can be imported into a program at any stage of model development.

2.2. Software architecture

The overall architecture layout of PlantGL is shown in Fig. 1, including several layers of encapsulation and abstraction. The geometric, algorithmic and GUI libraries lie in the core of PlantGL. The geometric library encapsulates a *scene-graph* data structure and a taxonomy of *geometric objects*. The *algorithm* library contains tools to manipulate and display geometric structures (for instance OpenGL rendering). The GUI is developed as a set of Qt widgets and can be combined with the previous components to provide visualization facilities. On top of this first layer, a set of wrappers create interfaces of the C++ classes into the Python language using the Boost.Python library [38]. Because we design our C++ libraries in an oriented-object manner, the C++ class interfaces were totally compatible with the Python framework. All these interfaces are gathered into a Python module named PlantGL. Additionally, some automatic conversion tools with standard python structures were added into the wrappers in order to reinforce compatibility and integration. This module is thus integrated seamlessly with standard Python and enables further abstraction layers, written in Python. Extension of the library can be done using either C++ or Python.

2.3. Basic components

The basic components of PlantGL are *scene-graphs* that contain scene objects (geometry, appearance, transformation, *etc.*), *actions* that define algorithms that can be applied to scene-graphs, and the *visualization tools*.

2.3.1. Scene-graphs

Scene-graphs are a common data structure used in computer graphics to model scenes [39]. Basically, a *scene-graph* is a directed, acyclic graph (DAG), whose nodes hold the information about the elements that compose the scene. In a DAG, nodes may have more than one parent, which enables parent nodes to share child nodes within the graph via the *instantiation* mechanism.

In PlantGL, an object-oriented version of scene-graphs was implemented to facilitate the customization of scene-graph node processing. Scene-graphs are made of nodes of different types. Types may be one of *geometry*, *appearance*, *shape*, *transformation*, and *group*. Scene-graph nodes are organized as a DAG hierarchy. Terminal nodes contain *shapes* pointing to both a geometry node (which contains a geometric model) and an appearance node (e.g. which defines a color, a material or a texture). Non terminal nodes correspond to grouping or transformation nodes that allow the user to set the position, orientation and size of a geometric object. They are used to build complex objects from simple geometric models.

Two families of geometric models are available in the library: *Explicit* and *Parametric* models. Explicit models are defined as sets of discrete primitives like points, lines or polygons that can be directly drawn by graphics cards. Parametric models offer a higher level of abstraction and are thus simpler to manipulate. However, to be displayed, parametric models have to be transformed into an explicit representation. The discretization process is explicitly controlled by parameters of the models that indicate how many discrete primitives have to be created.

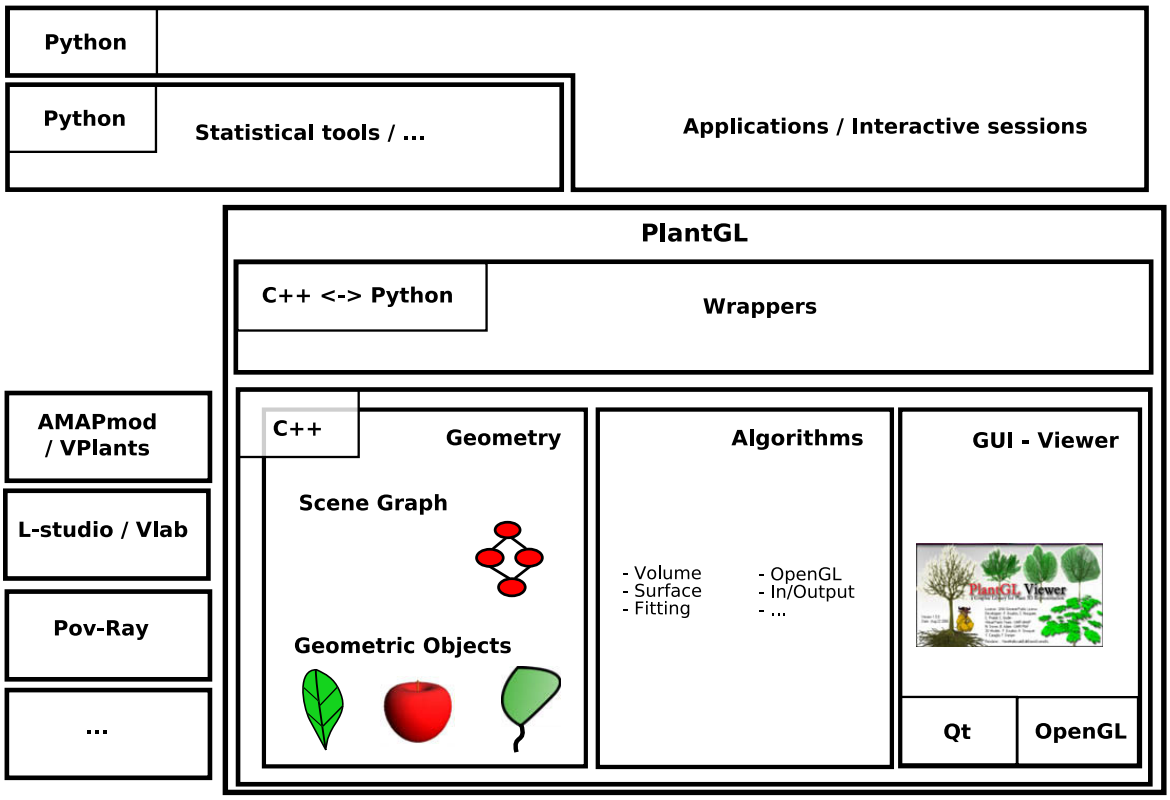

Fig. 1. Layout of the `PlantGL` architecture. It contains three C++ components: a geometric, an algorithmic and a GUI library. On top of this, wrappers implement interfaces with the `Python` language. `PlantGL` primitives can be used by modellers to develop scripts, procedures and applications in the embedding language `Python`. Data structures can be imported from and exported to other plant modelling software systems mentioned on the bottom left.

`PlantGL` contains a number of geometric models to represent points, curves, surfaces and volumes. They range from simple classical models such as *Cylinder*, *Sphere*, and *NURBS* to more specific ones, e.g. *Generalized Cylinder*, *Hull*, *etc.* Models dedicated to plant representation will be detailed in Section 3.

2.3.2. Algorithms

In `PlantGL`, algorithms are separated from data structures for flexibility and reuse purposes. Given the heterogeneity of the nodes contained in the scene-graph, an algorithm applied to a scene needs to adapt its execution according to the type of node it is applied to. For this, we implemented the *visitor* design pattern [37] that makes it possible to keep algorithms outside node objects definition by delegating the mapping from node to algorithms to a separate visitor object called an *action*. It is thus possible to add new algorithms by implementing new actions without modifying the node classes or loosing performance (more details in Appendix A).

In the actual implementation, `PlantGL` supports approximatively 40 *actions* that can be applied on a scene graph. The main ones can be classified into the following categories: *converters*, for instance from parametric to explicit representations; *renderers*; algorithms for the *characterization* of a scene using volume, surface or center of inertia; *fitting* algorithms to compute global representations from a set of detailed shapes using for instance bounding volumes (sphere [40], box), convex hull [41]; *ray casting* using CPU or GPU; *space partitioning* e.g. in an Octree; *etc.* This last structure make it possible to determine quickly spatial neighbours to test for possible intersections between geometries, for instance during organ positionning in plant model generation [42]. A family of algorithms also makes it also possible to build and export a scene in different scene description formats: some classical ones, such as `PovRay`, `Vrml`, *etc.* or of some plant dedicated systems like `AMAPmod/VPlants`, `LStudio/VLab`, `AMAPsim` and `VegeStar`. This reinforces interoperability of PlantGL with other modeling tools.

The use of these algorithms is illustrated in Section 4 in the context of different biological applications.

2.3.3. Visualization tools

The `PlantGL` *Viewer* (Fig. 2) provides facilities to visualize scene-graphs interactively. Different types of rendering modes are available including volumetric, wire, skeleton, and bounding box. Scene-graph organization and node properties can be explored with dedicated widgets allowing access to various pieces of information about the scene, like the number, volume, surface or bounding box of the scene elements. Simple editing features (such as a material editor) are available. Screen-shots can be easily made and exported to documents or assembled into movies. Most of the viewer features can be set using buttons or context menus and are also accessible procedurally.

The viewer and scenes are multi-threaded so that a user can manipulate a scene from a `Python` shell during visualization. Several types of dialog boxes can also be interactively created in the Viewer from the `Python` shell. Results

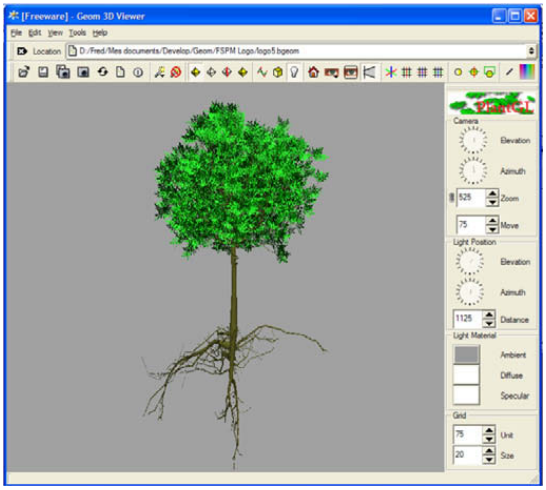

Fig. 2. Visualization of the detailed representation of a 15-year-old walnut tree [11] with the PlantGL viewer.

of the dialog are returned to the `Python` process using multi-threaded communications (see details in Appendix A). This enables graphical control of a `Python` modelling process through the Viewer.

2.4. Creating scene-graphs

There are different ways for application developers to create and process a scene-graph depending both on performance constraints and their familiarity with computer graphics concepts. First, `PlantGL` provides a declarative language, `GML` [43], similar to `VRML` [44] with extensions for plant geometric objects. `GML` mainly adds persistence capabilities to `PlantGL` in both ascii and binary versions. For `C++` applications, it is possible to link directly with the library. In this case, `PlantGL` features can also be extended by adding new objects and new actions. Additionally, the full `PlantGL API` is accessible from the `Python` language. Combination of high level tools and languages such as `PlantGL` and `Python` allows rapid prototyping of new models and applications, as illustrated in Section 4.

3. Construction of plant models

The design of `PlantGL` is focused on modelling and rendering of vegetative scenes. In this section, we present the specific geometric models and modelling tools of `PlantGL` useful for the representation of plant components at different scales. In an increasing computational complexity, we examine sequentially the representation of simple plant organs, tree crown, branching systems and cellular representation of tissues.

3.1. Plant organs

To represent simple organs, `PlantGL` contains a set of classical geometric models. For instance, some cylinders and frustums can be used to represent inter-nodes, NURBS patches to model complex organs such as leaves or flowers and generalized cylinders for branches and stems.

During the creation of a scene-graph, complex objects can be modeled using composition and instantiation. Simple procedures can be written in `Python` that position predefined symbols, for instance petals or leaves, using Transformation and Group objects, to create more complex objects by composition (Fig. 3). `Python`, as a modelling language, allows the user to express a full range of modelling techniques. For instance, the pine cone (Fig. 3a) is built by placing each scale using the golden angle [45]. The following code[2] sketches the placement of each scale.

```
from plantgl import *
pine = Scene ()
scale_smb = TriangleSet(...) # Geometric symbol of a pine scale
delta_angle = pi/(1+sqrt(5)) # golden angle between each scale
nb_scale, max_pine_radius, max_scale_size = 160, 50, 1.5
bottom_height, top_height = 10, 90 # global dimensions of the pine
def distToTrunk(u): # with u in [0,2], u < 1 for the bottom part
  if u < 1: return max_pine_radius*u
  else: return max_pine_radius*((2-u)**2)
def scaleSize(u):
  if u < 1: return max_scale_size*log(1 + u,2)
  else: return max_scale_size*log(3-u,2)
def scaleHeight(u):
  if u < 1: return u * bottom_height
  else: return bottom_height + (u - 1) * top_height
for i in range(nb_scale):
  u = 2*i/nb_scale
  pine += AxisRotated((0,0,1),i*deltaAngle,
    Translated((distToTrunk(u),0,scaleHeight(u)),
      Scaled(scaleSize(u),scale_smb))
```

In this example, pine scales are identified by a normalized u position along the trunk with $u \in [0,1]$ for the bottom part and $u \in [1,2]$ for the top part. For a scale at position u, the functions `distToTrunk`, `scaleSize` and `scaleHeight` give the distance to the trunk, its size and its height, respectively. Size of successive scales in the spiral follows a logarithmic law in this case. From this information and the golden angle, a geometric representation is built in the loop of the 5 last lines of code. Other algorithmic arrangements, such as the ones underlying Fig. 3b and c, can easily be made with Python.

3.2. Crown models

3.2.1. Envelopes

Tree crown envelopes are used in different contexts like studying plant interaction with its environment (i.e. radiative transfers in canopies [10] or competition for space [46]), 3D tree model reconstruction from photographs [7], and interactive rendering [47]. They are one original

[2] In all the code excerpts presented in this paper, standard python constructs are formated in bold to emphasize the structure of the code and the names of the structures and functions provided by `PlantGL` are underlined.

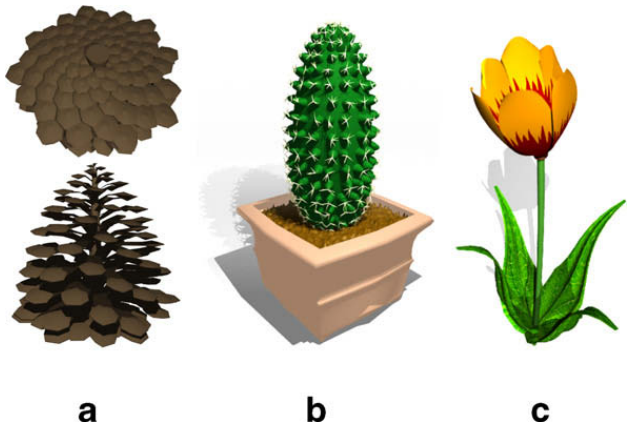

Fig. 3. (a) A pine cone, (b) a cactus, and (c) a tulip. Models were created with simple `Python` procedures that create and position organs, like petals and leaves.

key application of `PlantGL`. In this section, we describe in detail three envelope models that were specifically designed to represent plant volumes: asymmetric, extruded and skinned hulls.

Asymmetric hull: This envelope model, originally proposed by Horn [48] and Koop [49], then extended by Cescatti [10], makes it possible to easily define asymmetric crown shapes. The envelope is defined using six control points in six directions and two shape factors C_T and C_B that control its convexity (see Fig. 4). With a few easily controllable parameters, a large variety of realistic crown shapes can be achieved.

The first two control point, P_T and P_B, represent top and base points of the crown, respectively. The four other points P_1 to P_4 represent the different radius of the crown in two orthogonal directions. P_1 and P_3 are constrained to lie in the xz plane and P_2 and P_4 in the yz plane. These points define a peripheral line L at the greatest width of the crown. For the x and y dimensions, L is composed of four elliptical quarters. The height of points of L is defined as an interpolation of the heights of the control points (see Appendix B.1 for detailed equations).

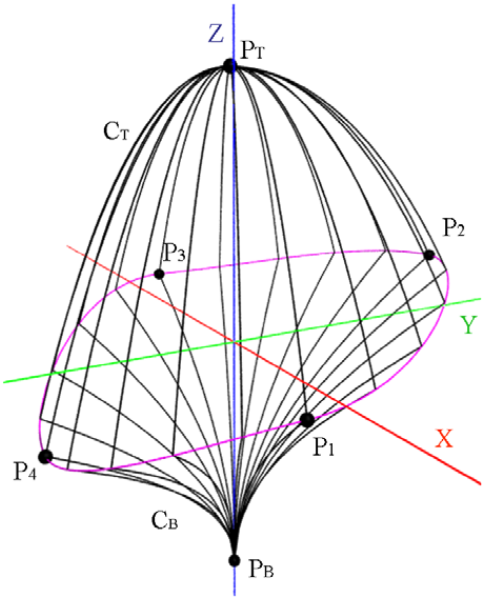

Fig. 4. Asymmetric hull parameters.

The two shape factors C_T and C_B describe the curvature of the crown above and below the peripheral line. Points of L are connected to the top and base points with quarters of super-ellipse of degrees C_T and C_B, respectively. Different shape factor values generate conical ($C_i = 1$), ellipsoidal ($C_i = 2$), concave ($C_i \in]0,1[$), or convex ($C_i \in]1,\infty[$) shapes. A great variety of shapes can be achieved by modifying shape factor values (see Fig. 5).

Cescatti proposed a simple methodology to measure the six control points and estimate the shape factors directly in the field. In `PlantGL`, a graphic editor has been implemented that makes it possible to build envelopes from photographs or drawings. For this, reference images can be displayed in the background of the different editor views. An illustration of this shape usage can be found for the interactive design of bonsai trees [30].

Extruded hull: The use of extruded envelope for plant modelling was originally proposed by Birnbaum [50]. Such an envelope is defined with a horizontal and a vertical profile. This first profile can be acquired from the projection of a tree crown on the ground, given for instance by shadow on the field; and the second one from a lateral view of the tree.

The envelope is reconstructed by sweeping the horizontal profile inside the vertical profile (see Fig. 6). For this, the vertical profile V is virtually split into slices by planes passing through equidistant points from the top (or by horizontal planes). These slices are used to map horizontal profiles inside V. Equations are given in Appendix B.2.

An illustration of the reconstruction of such an envelope from photographs is given in Section 4.1.

Skinned hull: The previous envelope models account for variable radius of the crown in different directions but have limited input allowing variation of the profiles shapes to be captured. To alleviate this difficulty, we propose a new flexible profile based envelope model, namely *skinned hull*, which is defined as the interpolation of a set of vertical profiles given for any angle around the z-axis.

The skinned hull is inspired from skin surfaces [51–53]. It generalizes the notion of surface of revolution with a variational section being an interpolation of the various profiles (see Fig. 7 and Appendix B.3 for equations). For the particular case of a single profile, the surface is a surface of revolution.

3.2.2. Foliage distributions

In particular studies such as light interception in ecophysiology or pest propagation in epidemiology, only the arrangements of leaves in space is relevant. This lead us to define adequate means to specify leaf distributions independently of branching systems.

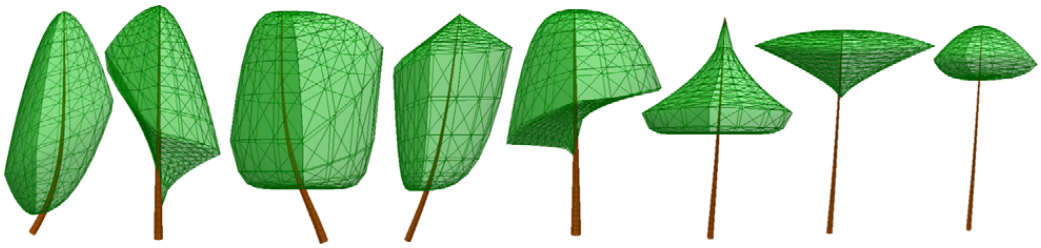

Fig. 5. Examples of asymmetric hulls showing plasticity of this model to represent tree crowns (inspired by Cescatti [10]).

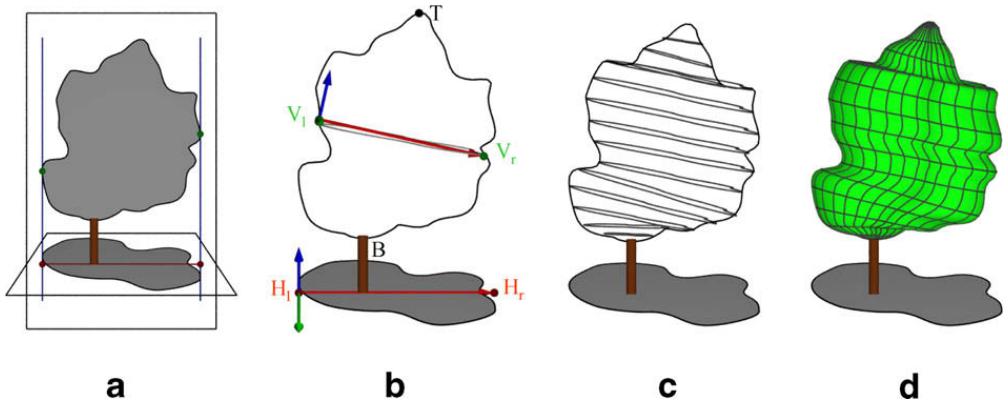

Fig. 6. Extruded hull reconstruction. (a) Acquisition of a vertical and a horizontal profile. (b) Transformation of the horizontal profile into a section of the hull. (c) Computation of all sections. (d) Mesh reconstruction.

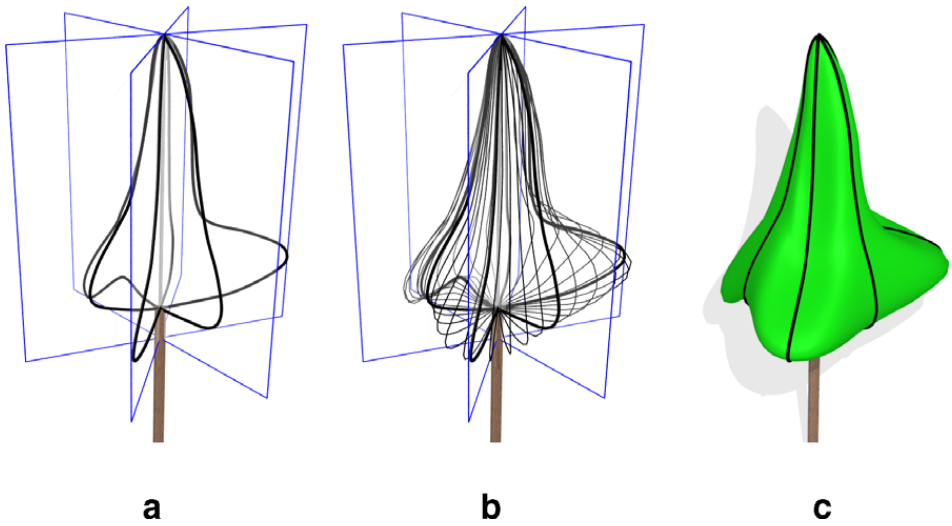

Fig. 7. Skinned Hull reconstruction. (a) The user defines a set of planar profiles in different planes around the z axis (in black). (b) Profiles are interpolated to compute different sections (in grey). (c) The surface is computed. Input profiles are iso-parametric curves of the resulting surface.

A first simple strategy consists of creating random foliage according to different statistical distributions. This can be easily expressed in `PlantGL` as illustrated by the following example. Let us consider a random canopy foliage whose leaf distribution varies according to the height in the canopy. The following code sketches the creation of three horizontal layers of vegetation with different statistical properties. In this example, leaves are generated above an horizontal rectangle delineated by `(xmin, xmax)` and `(ymin, ymax)`. Layers L_i are positioned between bottom and top heights (e.g. between `heights[`$i-1$`]` and `heights[`i`]`). To simulate different leaf distributions in the foliage, we associate an increasing probability to each height for a leaf to be above this height, bottom height of first layer having $p_0 = 0$ and top height of last layer having $p_3 = 1$. For the three layers, a leaf will thus have a probability p_1 to be in the first layer, $p_2 - p_1$ in the second layer, and $1 - p_2$ in the third one.

```
from random import uniform
sc = Scene ()
heights = [ 1, 2, 3, 4 ] # Height of the layers limits
leaf_symbol = TriangleSet(...) # Leaf geometry to
   instantiate
for i in range(nbleaves):
   p = uniform (0,1)
   if p <= p1:
      height = uniform (heights[0], heights[1])
   elif p1 < p <= p2:
      height = uniform (heights[1], heights[2])
   else:
      height = uniform (heights[2], heights[3])
# random position and orientation of leaves at the
   chosen altitude
pos = (uniform (xmin, xmax), uniform (ymin, ymax),
   height)
az, el, roll = uniform(−pi, pi), uniform(0, pi),
   uniform(−pi, pi)
sc += Translated (pos, EulerRotated (az, el, roll,
   leaf_symbol)))
```

Fig. 8 shows the resulting random foliage (trunk generation is not described in the code).

Plant foliage may also exhibit specific leaf arrangement with regular and deterministic properties. If the regularity corresponds to a form of spatial periodicity at a given scale, a procedural method similar to the previous one for random foliage can be easily used. However, some plants like ferns or pines show remarkable spatial organization of their foliage with several levels of aggregation and similar structures repeated at the different scales [15]. Such fractal spatial organizations can be captured by 3D *Iterated Function Systems* (IFS) [54].

An IFS is a set of contracting affine transformations $\{T_i\}_{i\in[1,M]}$. This family of contractions defines a contracting operator F for all bounded set of points X in $\mathbb{R}^3$ such that:

$$F(X) = T_1(X) \cup T_2(X) \cup \ldots \cup T_M(X). \quad (1)$$

The F operator may be applied iteratively to an initial arbitrary 3D geometric model, I, called the *initiator*. At the nth iteration the obtained object L_n has a pre-fractal structure composed of a number of elements increasing exponentially with n (while the size of each element decreases at an equivalent rate).

$$L_n = F^n(I) \quad (2)$$

When n tends to infinity, the iteration process tends to a fractal object L_∞, called the attractor. The attractor only depends on F (and not on the initiator) and has a known fractal dimension that depends on the contraction factors and number of F transformations, e.g. [55].

IFSs have been implemented in `PlantGL` as a transformation primitive. They may be used to construct reference virtual plant foliages with *a priori* determined self-similar structures and dimensions, e.g. [15,34]. The following code shows the construction of the IFS of Fig. 9. The affine transformations are defined as 4x4 matrices that are build here from a translation (`t`), a scaling factor (`s`) and a rotation defined from some euler angles (`a`). The initiator is a disk that represents a simple leaf shape.

```
transformations = [Matrix4.fromTransform (t, s, a) for t,
   s, a in
((0, 0, 2), 1/3, (0, 0, 0)), ((0, 0.5, 1.5), 1/3, (45, 90, 0)),
((0, -0.5, 1.5), 1/3, (45, 270, 0)), ((0.5, 0, 1), 1/3, (45, 0,
   0)),
((-0.5, 0, 1), 1/3, (45, 180, 0)), ((0, 1, .5), 1/3, (45, 90, 0)),
((0, -1, .5), 1/3, (45, 270, 0)), ((1, 0, 0), 1/3, (45, 0, 0)),
((-1, 0, 0), 1/3, (45, 180, 0))]]
i = IFS(4, transformations, Disk(1))
```

3.3. Branching system modelling

3.3.1. Scene-graphs for plants

In `PlantGL`, the construction of a branching system comes down to instantiating a *p-scene-graph*. A p-scene-graph corresponds to an adaptation of the PlantGL scene-graph to the representation of plant branching structures. For this, a particular set of nodes in the p-scene-graph, named structural nodes, are introduced and represent the different components of the plant. These nodes are organized as a tree graph (as described in [56]) in which two types of edges can be specified to distinguish branching (+) and succession (<) relationships between parent and child nodes (see Fig. 10a). In addition, each structural node is connected with transformation, geometry and appearance nodes that define the representation of each component. In p-scene-graphs, transformations can either be specified in an absolute or relative mode. In the absolute mode, transformations are expressed with respect to a common global reference frame whereas, in the relative mode, transformations are expressed in the reference frame of the parent component in the tree graph.

Fig. 8. A layered canopy foliage. The probabilities for a leaf to be in the first, second or third layer are, respectively, 0.1, 0.7 and 0.2 (giving $p_1 = 0.1$ and $p_2 = 0.8$).

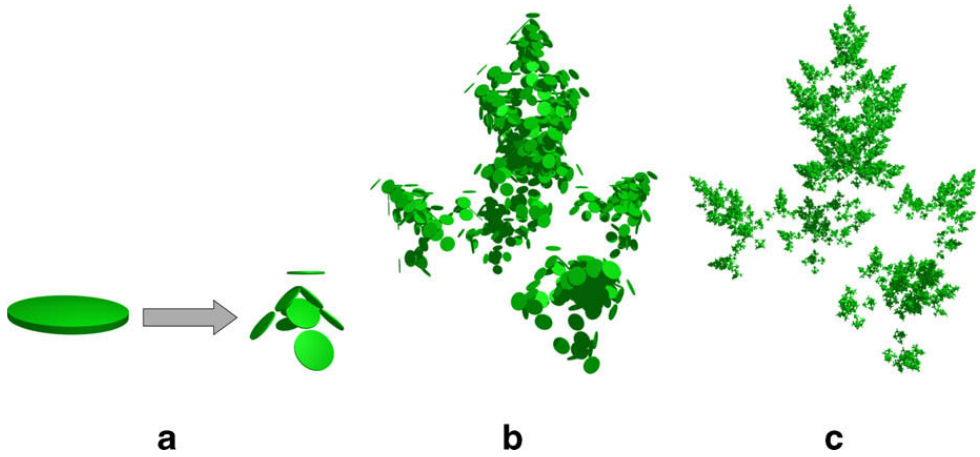

Fig. 9. Construction of a fractal foliage using an IFS. (a) The initiator is a disc (representing a leaf shape). In the first iteration, the initiator is duplicated, translated, rotated and resized according to the affine transformations that compose F, leading to L_1. (b and c) IFS foliages L_3 and L_5 at iteration depths 3 and 5. The theoretical dimension of this foliage is 2.0.

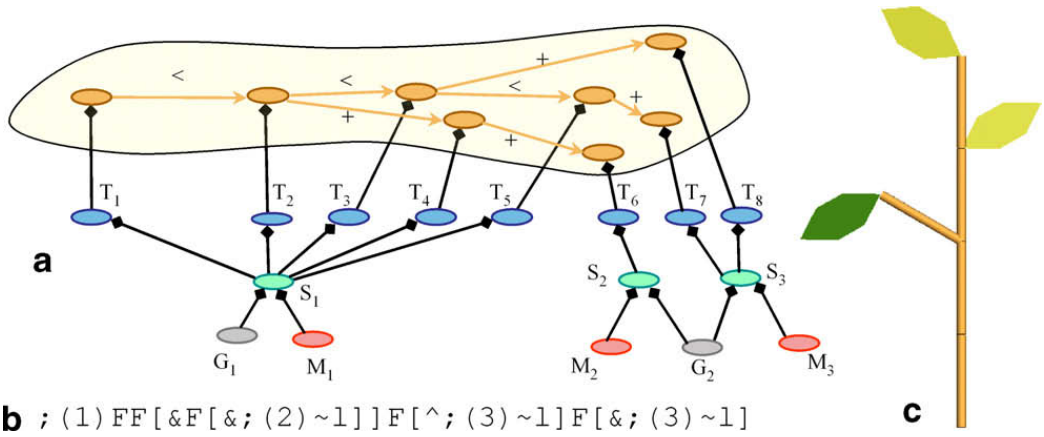

Fig. 10. A p-scene-graph. (a) The scene-graph with structural nodes in orange, transformation in blue, shape in green, appearance in red and geometry in grey. (b) The corresponding L-systems bracketed string from which it has been constructed. The '`F`' symbols are geometrically interpreted as cylinders, '`~l`' as leaf symbols, '&' and '^' as orientation operations and ';' as color specifications, [57]. (c) The corresponding geometric model. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The topological relationships specified in the p-scene-graph express physical connections between plant components. For this, a set of constraints, namely *within-scale constraints*, can formalize these connections in term of geometry [26]. These constraints may specify for instance continuity relationship between the geometric parameters of connected components (end points, diameters, etc.). These constraints are used to ensure the consistency of the overall representation.

P-scene-graphs are further extended by introducing a multiscale organization in the structural nodes, which makes it possible to augment the multiscale tree graphs (MTG) used in plant modelling [56] with graphical informations [58]. Such multiscale graphs correspond to recursively quotiented tree graphs [56]. It is possible to define multiscale organization from a simple detailed tree graph (namely the support graph) by specifying quotient functions that will partition the nodes into groups corresponding to macroscopic components in the MTG.

The different scales included in the multiscale p-scene-graph give different views with different resolutions of the same plant object. Since they correspond to different views of the same reality, the associated geometric models must respect particular coherence constraints. For this, a set of *between-scale constraints* is defined that relate the model parameters at one scale with the model parameters at other scales. Between-scale constraints may specify for instance that all the components of a branching system must be included in some macroscopic envelope. These constraints may be either used in a bottom–up or top–down fashion. In top–down approaches, macroscopic representation may be used to bound the development of a plant at a more microscopic level. Such a strategy was used for instance in [30] for the design of bonsai tree using L-systems. In bottom–up approaches, a detailed representations of a plant is used to compute a more macroscopic representation. For this, a set of fitting algorithms has been implemented in `PlantGL` that makes it possible to compute the bounding envelope of a set of

geometric primitives using for instance convex hulls [41] or minimal bounding sphere [40], *etc*. These envelope representations can thus be used to characterize globally the geometry of a branching system at different scales, for instance for computing the fractal dimension of a plant [15].

3.3.2. Construction of branching structure models

P-scene-graphs can be created either by generative procedures written in Python or by importing plant structures from other plant modeling software.

From a generative perspective, we particularly emphasized in the current version of PlantGL the connection with L-studio/VLab [6], a widely used L-system based modelling framework for plant growth modelling. An L-system [3] is a particular type of rewriting system that may be used to simulate the development of a tree like structure. In this framework, the tree structure is encoded as a bracketed string of symbols called modules that represent components of the structure. To represent attributes, these modules may bear parameters. Particular bracket symbols mark the beginning and the end of branches. The plant growth is formalized by a set of production rules that describes the change over time of the string modules. Starting from an initial string, namely the axiom, modules of the string are replaced according to appropriate rules. A derivation step corresponds to the rewriting in parallel of every module of the string. Therefore, the development of a tree structure through time is modelled by a series of derivation steps. To associate a geometric interpretation with the L-system's output string, some modules are given a graphical meaning and a LOGO-style turtle is used to interpret them, [59]. For this, the string is scanned sequentially from left to right and particular modules are interpreted as actions for the turtle. The turtle state is characterized by a position, a reference frame and additional graphical attributes. Some modules make the turtle move forward and draw graphical elements (cylinders, *etc*.) or change its orientation in space. A stack mechanism makes it possible to store the turtle state at the beginning of a branch and to restore it at the end.

In order to interface L-studio/VLab with PlantGL, import procedures have been implemented in PlantGL. The strings representing the branching systems generated with cpfg are stored in text files. These strings are then imported into PlantGL with the dedicated primitive Lstring. To interpret the L-system modules as commands for the creation of a p-scene-graph, a particular turtle has been implemented in PlantGL that follows the Lstudio/VLab specification [3] (see Fig. 10). Since the p-scene-graph is accessible in Python, it is then possible to interactively explore and analyse the resulting geometric structure with the set of algorithms available in PlantGL for the manipulation of a scene-graph or for the analysis of a plant topological structure using other toolkits such as AMAPmod/VPlants [60]. The following code sketches the import of a L-system string in PlantGL, its conversion into a scene-graph and some basic manipulation of the result such as display and wood volume computation.

```
lstring = Lstring ('plant.str')
turtle = PglTurtle ()
lstring.apply (turtle)
sg = turtle.getSceneGraph ()
Viewer.display (sg)
vol = volume (sg)
```

A more complex example of such a coupling between Lstudio/cpfg and PlantGL is illustrated in Section 4.3.3. It uses PlantGL's hulls defined in Section 3.2.1 to constrain L-systems generation of branching structure. Other connections with modelling platforms such as AMAPmod/VPlants [26], and VegeSTAR [61] are also available and make it possible to create 3D plant models from measured data (see Fig. 11).

3.4. Tissue models

In PlantGL, a plant tissue is considered as a collection of connected regions. A region may represent either a single cell or a set of cells. Unlike branching systems, the

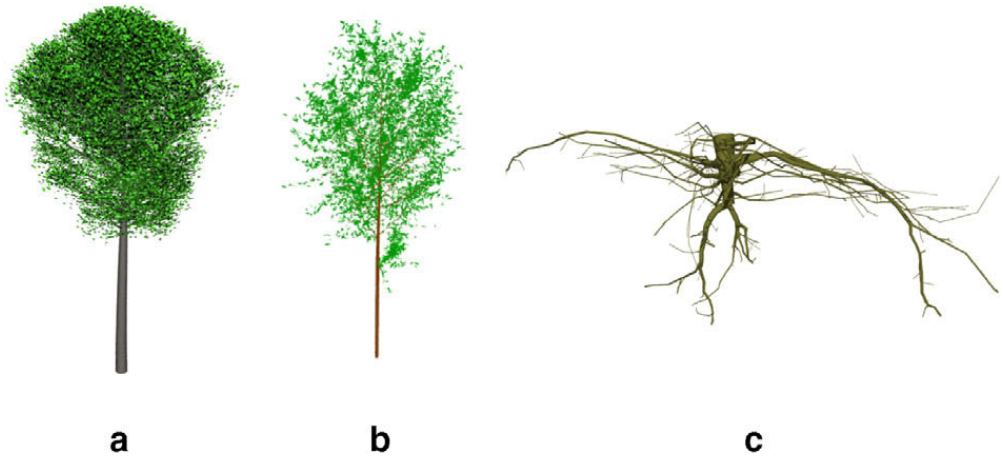

Fig. 11. Example of branching systems in PlantGL. (a) A beech tree simulated with an L-system using Lstudio/VLab and imported in PlantGL. This model is used in the application presented in Section 4.3.3. (b) Black tupelo tree generated procedurally in Python using PlantGL according to the generative procedure proposed by Weber and Penn in [4] and (c) the root system of an oak tree [13].

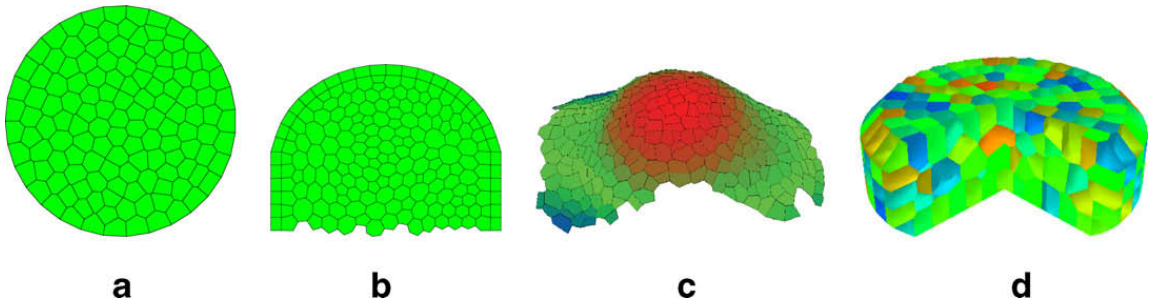

Fig. 12. Tissue models. (a) 2D tissue, (b) 2D transversal cut, (c) 3D surface tissue from [33], (d) 3D tissue.

neighborhood relationship between regions cannot be simply represented by tree graphs since the connection networks between regions usually contain cycles. To model the neighborhood relationship between regions correctly, we need to take into account the hierarchical organization of region connections: for example, in 3 dimensions, two 3D cells are connected through a 2D wall, two walls are connected through a 1D edge and two edges are connected through a 0D vertex. More generally, the connection between two or more elements of dimension $n+1$ is an element of dimension n. Such a hierarchical organization defines an *abstract simplicial complex* [62]. Similarly to *p-scene-graphs* for branching systems, scene-graphs representing tissues, called *t-scene-graphs*, are defined by augmenting simplicial complexes representing cell networks with geometrical properties. Each structural node of the simplicial complex is associated with transformation, geometry and appearance nodes (Fig. 12).

A set of geometric algorithms has been designed to simplify the manipulation of the *t-scene-graphs* during simulations.

- The first algorithm makes it possible to define the geometry of an element of dimension $n+1$ from the geometric information of its components of dimension n. For example, the polyhedral geometry of a cell is derived from the polygonal geometry of its walls. The overall consistency of the geometry of all elements in the tissue is thus ensured by specifying only the geometry of its smallest elements.
- The second algorithm implements cell division. A cell (or more generally a region) is divided in two daughter cells. The geometry of these daughter cells may be specified by the user using standard `PlantGL` algorithms (that compute main axis, shape volume or surface, shape orientation, ...) to reflect the biological characteristics of cell division geometry (main orientation of the cell, smallest separation wall, orthogonality between walls, ...).
- The third algorithm has been designed to refine *t-scene-graphs* into small triangular elements. Resulting meshes can be used either to visually display the tissue or in conjunction with finite element methods to solve differential equations representing physiological processes (diffusion, reaction, transport, ...) or mechanical stresses for example.

T-scene-graphs can be obtained either from a file, from images of biological tissues [33], or using procedural algorithms. `PlantGL` provides a set of procedural algorithms that generate regular, grid-based tissues (based on rectangular or hexagonal grids) and non-regular tissues containing cells with random sizes. Random tissues are generated using a randomly placed set of points representing cell centers. The Delaunay triangulation (2D or 3D) of this set of points is then computed. This is done by using an external computational geometry library, `CGAL` [63], available in `Python`. Cell neighborhood is defined by this triangulation and walls correspond to its dual representation (Voronoï diagram). These procedural algorithms result in relatively simple tissue structures. More complex *t-scene-graphs* can be obtained by simulation of tissue development. Starting from an initial simple tissue, a growth algorithm modifies the shape of the tissue. A cell is divided each time its volume reaches a given threshold. This combination of growth and division is maintained up to the desired final shape (see 4.3.1 for example).

4. Applications and illustrations

The `PlantGL` library has already been used in a number of modeling applications by our group (e.g. [33,34,15,31,36]) and other plant research groups (e.g. [32,35]). The library allows modellers to address graphic and geometric issues in the different phases of a modeling process, i.e. observation, analysis, simulation and model evaluation. In this section, we aim to illustrate how `PlantGL` provides a set of useful efficient tools to address various questions in these different phases. In particular, we stress the use of envelope- or grid-based approaches which is original in `PlantGL` and opens new application areas. In these applications, we illustrate how `PlantGL` can be assembled with other `Python` libraries to achieve high-level operations on plant structures, thus opening the way to the definition of a powerful plant modeling platform.

4.1. Plant canopy reconstruction

In plant modeling, 3D digitizing of plant structure has become a topic of increasing importance in the last decade. Various methodologies have been used to digitize plants at different levels of detail for leaves, for branching systems, and also for tree crowns [64,65,26,50,7,46]. Among these approaches, the reconstruction of 3D models of large/tall trees (like trees of a tropical forest for example) remains a challenging problem. This is mainly due to the difficulty

of acquiring information in the field, and in capturing the intricate structure of such plants. In this section, we show how the new envelope-based tools provided in PlantGL can be used for this aim.

4.1.1. Using PlantGL to build-up large crowns

First approaches attempted to use parametric models to estimate the geometry of tree crowns in the context of light modeling in plant stands [10]. More recently, non-parametric visual hulls have been used in different application contexts to characterize the plant volume based on photographs [7,47,46].

PlantGL makes it possible to easily combine these approaches by using for example photographs and parametric envelope models to estimate plant canopy volumes.

Using a set of images of a tree (being either photographs or botanical drawings) with known camera positions and orientations, the modeler has to define a number of crown profiles according to the selected envelope model. For the extruded hull for instance, two closed curves that encompass the entire crown in vertical and horizontal planes are required. This step is usually made by manually outlining the foliaged tree region using an internal curve editor. Profiles are then used to build the tree silhouette hull.

From this estimated crown envelope, the modeler can then infer a detailed crown model by making assumptions about the leaf distribution inside the crown. Fig. 13 illustrates the use of a simple uniform random distribution of leaf positions and orientations. The following code shows how the example for random foliage generation of Section 3.2.2 can be adapted to take into account complex crown shapes.

```
hull # reconstructed hull
bbx = BoundingBox (hull)
foliage = Scene ()
def random_position ():
  return (uniform (bbx.xrange ()),uniform (bbx.yrange
  ()),
    uniform (bbx.zrange ()))
for i in range (nbleaves):
  pos = random_position ()
  while not inside (hull,pos):
   pos = random_position ()
foliage += Translated (pos,leaf_symbol))
```

Using this code, the generation of the foliage of Fig. 13c composed of 2000 leaves is made in 1 s on a Pentium IV 2.0 GHz.[3] Using Python, it is possible to define foliage distribution using more complex algorithms, such as those described in [6,7,30].

4.1.2. Using PlantGL to assess plant mock-up accuracy

Another important problem in canopy reconstruction is to assess the accuracy of 3D plant mockups obtained from measurements. A family of solutions consists of comparing equivalent synthesized descriptions of both the real and the virtual plants. In this family, hemispherical views are particularly interesting since they directly measure a physical characteristic of the plant, namely the amount of intercepted light.

Fig. 14 illustrates this approach [66]. A hemispheric picture is taken from the real plant, while an equivalent virtual picture is computed with the same camera position on the reconstructed plant. White areas in both pictures directly reflect the amount of light that reach different positions under or inside the crown [67]. The amount of intercepted light is summarized with the *canopy openness index* defined as the ratio between white pixels and total number of pixels on the picture.

4.2. Analysis of plant geometry

Plant geometry is a parameter of paramount importance in the modeling of plant-environment interactions. However, plants usually show complex geometric shapes with numerous components, highly organized but with non-deterministic structure. Characterizing this "irregularity" of plant shapes with few high level parameters is thus a determinant issue of modeling approaches. In many applications in forestry, horticulture, botany or eco-physiology, analysis of plant structures are carried out to find out adequate ways of capturing their intricate geometry in simple models. In this section, we illustrate the use of grids and envelopes defined in PlantGL in order to achieve such analysis.

4.2.1. Grid-based analysis

Fractal geometry was introduced to analyse the geometry of markedly irregular structures that can be either mathematically constructed or found in nature [69]. Several parameters have been introduced for this purpose, such as fractal dimension and lacunarity. These parameters are intended to capture the essence of irregularity, i.e. the way these structures physically occupy space as resolution decreases. Several estimators of these parameters exist. They consist of paving the original object in different manners with tiles of different sizes and studying the variation of the number of tiles with tile size.

For plant structures, fractal properties, such as fractal dimension, have been computed in different contexts, e.g. [71]. They frequently rely on the fractal analysis of 2D photographs. However, more recently, several works showed the possibility to compute more accurate 3D-estimators using detailed 3D-digitized plant mock-ups of real plants [72,15,34].

PlantGL makes it possible to carry out such computation in a flexible way. For example, to implement the box-counting estimator of the fractal dimension [69], the PlantGL "grid" object can be used to count the number of 3D cells of a given size containing vegetation. If $N(\delta)$ denotes the number of occupied 3D cells of size δ, the box-counting estimator of the fractal dimension D_δ of the object is defined as:

$$D_\delta = \lim_{\delta \to 0} \frac{\ln N(\delta)}{\ln \frac{1}{\delta}}. \quad (3)$$

[3] The computation times indicated in the remainder of this paper are for the same hardware configuration.

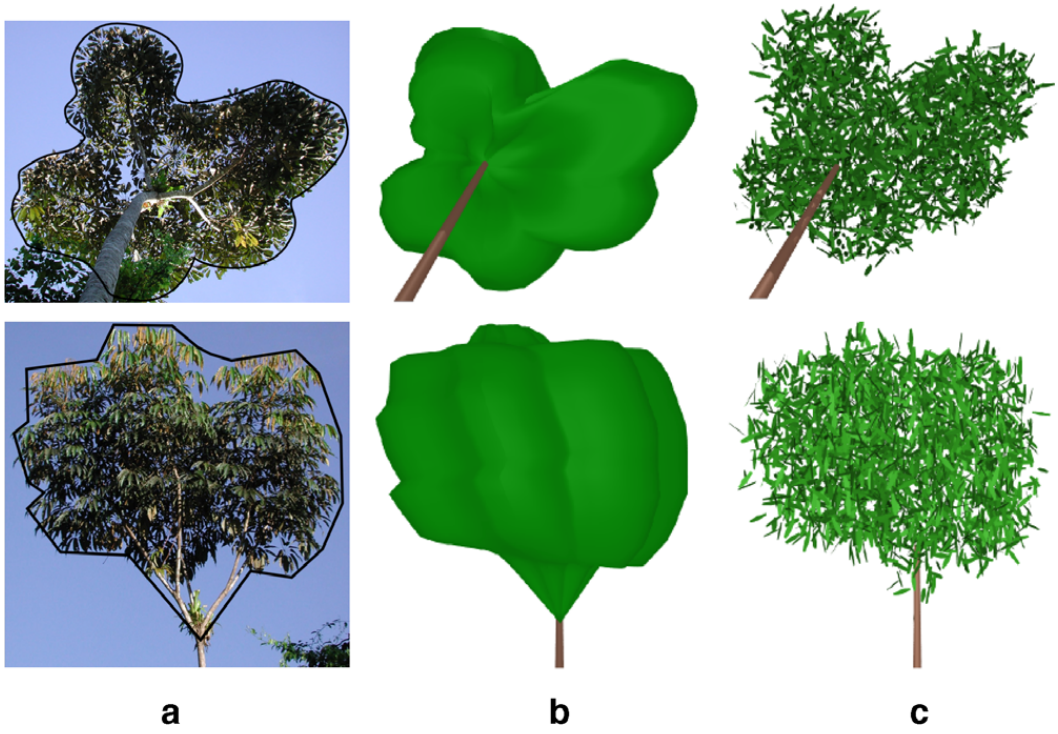

Fig. 13. Reconstruction of the crown envelope of a *Schefflera Decandra* tree with an extruded hull built from two photographs. Sketches are done with an internal curve editor. Photo courtesy of Y. Caraglio.

Fig. 14. Assessment of plant model reconstruction using hemispheric view [66]. The *Juglans nigra* × *Juglans regia* hybrid walnut plant model is reconstructed from partial measurements: wood structure is manually digitized while leaves are generated from distribution functions. On the left, an hemispheric photograph of a real walnut from the ground. On the right, reconstructed mockup using `AMAPmod/VPlants` exported to `Pov-Ray` [68] to compute a hemispheric picture at the same position. Here, the evaluated canopy openness is 40% for the real photograph and 49 % for the virtual tree.

D_δ is estimated from the slope of the regression between $\ln N(\delta)$ and $\ln \frac{1}{\delta}$ values. The following code sketches the implementation of such an estimator using `Python` and `PlantGL`.

```
from scipy import stats
def boxcounting (scene,maxdivision):
nbvoxels = [log (Grid (scene, div).nb_intercepted_voxels ())
            for div in range (maxdivision)]
delta = [log (1./i) for i in range (maxdivision)]
slope, itcept, r, ttp, stderr = stats.linregress (nbvoxels, delta)
return slope # slope of the regression
```

Fig. 15 illustrates the application of the box counting method on the foliage of a 3D-digitized apple tree [70]. For this example, computation of the 30 grids of decreasing sizes took 0.7 s. `PlantGL` can be used in a similar way to compute various fractal properties of plants [15,34]

4.2.2. Envelope-based analysis

Parametric envelopes provided in `PlantGL` can also be used to analyse volumetric properties of plant crowns. For example, in order to quantify the development of a plant crown over time, envelopes can be adjusted to the crown of the developing tree at different ages and their surface or volume can then be estimated.

Fig. 16 illustrates this approach together with the possibility to import plant data from other software. The growth of a eucalyptus was simulated at various ages using the `AMAPsim` software [23], Fig. 16a. Results were imported

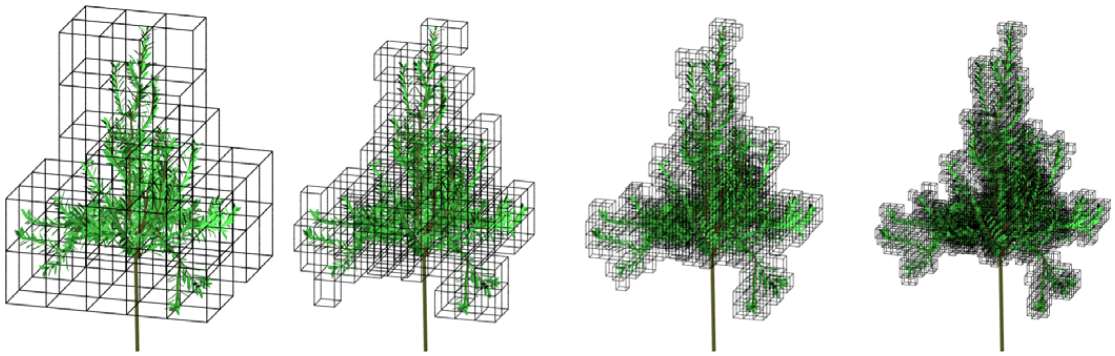

Fig. 15. The box counting method applied to the foliage of a digitized apple tree [70]. The tree is 2 m height with around 2500 leaves. Global bounding box has a volume of 10 m^3 and serves as the initial voxel. A grid sequence is then created by subdividing uniformly this bounding box into sub-voxels. At each scale, intercepted voxels are counted to determine fractal dimension (here of the order of *2.1*).

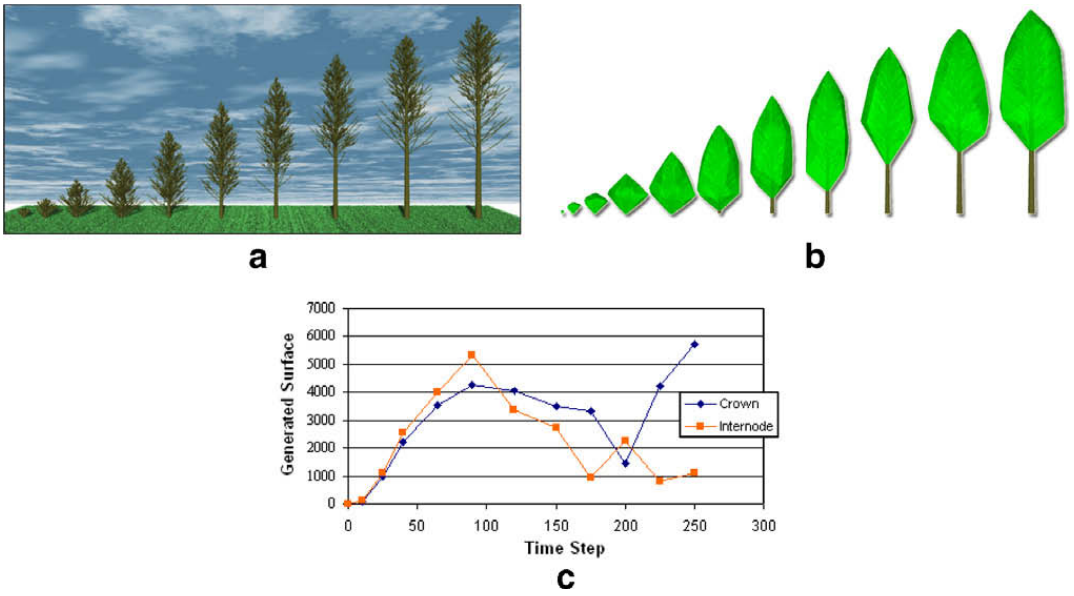

Fig. 16. (a) A eucalyptus tree simulated at various age (from 1 to 8 months on a scale from 10 to 250) with the `AMAPsim` software, exported to PlantGL and rendered with `Pov-Ray` software [68]. (b) Global representations of eucalyptus crown at the various ages. (c) Increments of wood and hull surfaces over time. The different degrees of correlation and their associated time period enable us to identify the various phases of the crown development.

in `PlantGL` as MTGs. The convex hull of the plant crown was then computed at each age using the fitting algorithms provided by `PlantGL` for envelopes. Here is how this series of steps can be carried out in `PlantGL`:

```
import pylab
def hullanalysis (ages, trees):
hulls = [fit ('convexhull', i) for i in trees]
wood_sf = [surface (i) for i in trees]
hull_sf = [surface (i) for i in hulls]
delta_wood_sf = [wood_sf[i+1]-wood_sf[i] for i in range
   (len (wood_sf)-1)]
delta_hull_sf = [hull_sf[i + 1]-hull_sf[i] for i in range
   (len(hull_sf)-1)]
pylab.plot(ages[1:], delta_wood_sf)
pylab.plot(ages[1:], delta_hull_sf)
```

Based on such data, various investigations about the crown development can be made. Curves showing the variation of crown surface/volume through time can be analysed as shown in Fig. 16c. Comparison at a more microscopic scale with the leaf area variation can thus be made.

4.3. Simulations based on plant geometric models

The use of flexible geometric models of plants is not restricted to the analysis of plant structure. They can be used as well for the simulation of various physical or physiological processes that take place *within* or *in interaction with* the plant structure. Here, we present three applications that demonstrate the use of `PlantGL` at different scales, ranging from organ to communities.

4.3.1. Simulation at organ scale

Due to the recent advances in plant cellular and developmental biology, the modeling of plant organ development is considered with a growing interest by the plant research community: leaf [73–75], shoot apical meristem [76–80], and flower [81]. PlantGL provides flexible data

structures and algorithms that make it possible to develop 2D or 3D simulations of tissue development.

As a matter of illustration, let us model the development of a bump formation in a tissue for instance to simulate development of a primordium at a shoot apex. We assume that the tissue is composed of polygonal cells in 2D (respectively polyhedral cell in 3D) delimited by 1D walls in 2D (respectively by polygonal walls in 3D).

The bump formation of a primordium at the surface of a meristem can be approximated (in cylindrical coordinates) by:

$$z(r,t) = \frac{h(t)}{1 + e^{k(r - r_m(t))}} \quad (4)$$

where h is the height of the bump, r_m is the radius of the bump at half its height and k is a shape factor that defines the slope of the bump. h and r_m vary throughout time at a rate ρ_h and ρ_r, respectively. From this equation we can derive a time dependent velocity field for surface vertices. The velocity of the internal vertices is an interpolation between the surface velocity and the velocity of the most inner vertices. To simplify, we assume that the most inner vertices (at altitude z_{min}) are fixed and their speed is then equal to zero. At each time t of the simulation, each vertex is moved according to its velocity. This process progressively modifies the cell size, and consequently the overall tissue shape. During the growth, if a cell has a volume (or surface in 2D) that reaches a predefined threshold, it divides into two children cells. Different algorithms implementing cell division are available on a tissue object, *e.g.* [82]. Here follows the code of such a tissue growth in `PlantGL` for a particular choice of the cell division algorithm:

```
tissue = createGridTissue ( (20, 5) )
def surface_altitude (r, t):
  return h(t)/(1 + exp(k*(r-rm(t))))
def velocity_field (pos, t):
  r = norm (pos.x, pos.y)
  tmp = exp(k*(r-rm(t))
  surface_speed = (rho_h(t) + k*h(t)*rho_r(t)*tmp/(1
  + tmp))/(1 + tmp)
  zspeed = surface_speed*(pos.z-zmin)/
  (surface_altitude(r,t)-zmin)
    return Vector3(0, 0, zspeed)
for t in range(time_begin, time_end, delta_time):
  for pos in tissue.positions () :
  pos += velocity_field(pos, t)*delta_time
for cell in tissue :
  if cell.size () > max_cell_size:
    cellDivide (cell, algo = MAIN_AXIS)
```

Results of the simulation are presented on Fig. 17. This growth simulation of the tissue, composed of 100 cells, using 100 time iterations took 4.5 s. Note that interpolation has been slightly modified to account for a constant size of the two external layers of cells (see Fig. 18).

4.3.2. Simulation at plant scale

In biological applications, virtual plants are frequently used to carry out virtual experiments where data is difficult to measure or when the interaction between the studied processes is too complex. This is particularly true for the study of light interception by plants: light cannot be measured in a real canopy with high accuracy and the amount of light rays that can go through a canopy is a complex function of the tree architecture. While the canopy

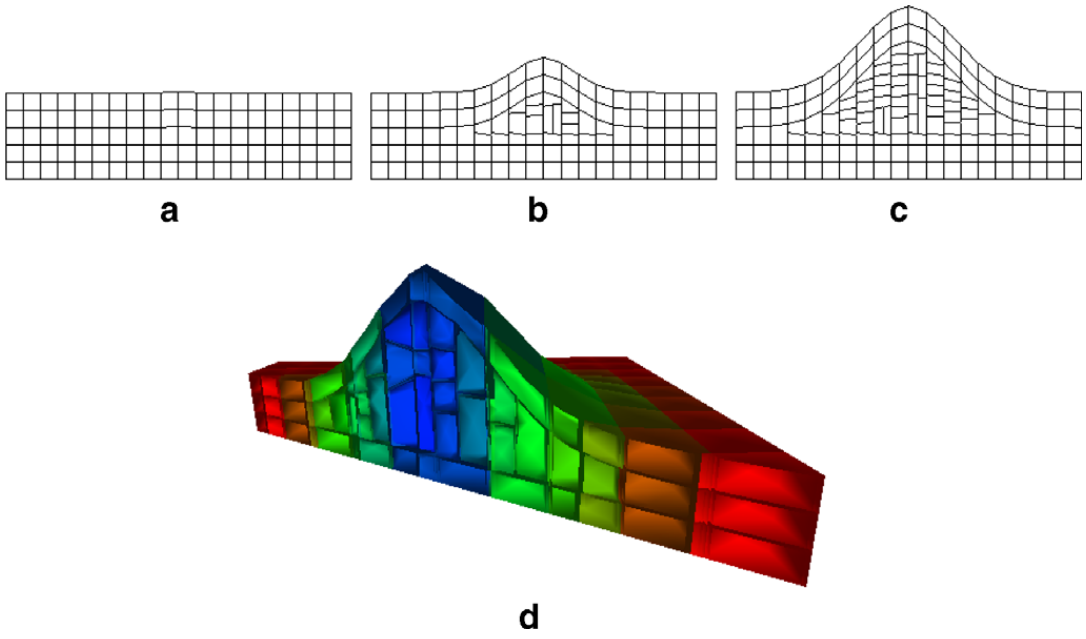

Fig. 17. (a–c) 2D geometrical representation of a growing tissue at three different times. (d) 3D simulation of bump formation on a tissue.

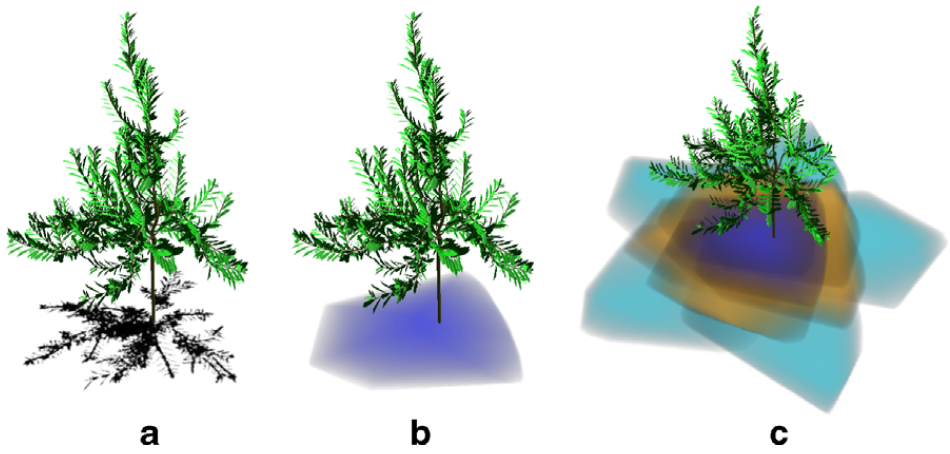

Fig. 18. Light intercepted by an apple tree represented as shadow on the ground. Intensity of the colors represent intensity of interception. STAR can be computed as a ratio between the area of the shadow and the plant leaf area. (a) Light intercepted from top direction using ray casting. STAR in this direction is equal to 0.23. (b) Light intercepted from same direction using Beer–Lambert hypothesis. STAR in this case is equal to 0.34. (c) Light interception sampled from different directions. The different colors are used to mark the difference between various elevations of ray direction. Sky integrated STAR is equal in the case of ray casting to 0.44 and in the case of Beer-Lambert to 0.58. This confirms that the turbid medium hypothesis over-estimates the STAR measure [86].

openness index presented in Section 4.1.2 gives an estimation of the ratio of light intercepted by the plant from one view point, this example addresses the problem of integrating such a measure on the whole plant structure. Additionally, it illustrates the use of `PlantGL` in the context of model assessment and shows how high-level geometric operations used in light interception models can be simply performed with `PlantGL`.

Light intercepted by a plant can be characterized by STAR values, namely Surface to Total Area Ratio [83]. This eco-physiological parameter is a directional quantity defined by taking the ratio of the area of the projection of a tree foliage S_Ω in a particular direction Ω to its total leaf area S. The STAR is thus the mean irradiance of a leaf area unit.

$$STAR_\Omega = \frac{S_\Omega}{S} \tag{5}$$

The directional STAR index can be integrated over all the sky vault to characterize the overall light interception of a tree. For this, a set of directions given by the turtle sky discretization [84] can be used and associated STAR values are averaged after weighting by the standard overcast sky radiance distribution [85].

Since the total leaf area of a real plant is often expensive to measure, approximate values of the STAR are often used in eco-physiological applications. For this, the directional STAR is estimated from simple measures of the plant volume and leaf density and by making simplifying assumptions on the actual spatial distribution of leaves in the canopy [86]. In this case, the plant is supposed to be a homogeneous volume with small leaves uniformly distributed within the crown looking like a "turbid medium". In this context, a light beam b of direction Ω_b has a probability $p_0(b)$ to be intercepted:

$$p_0(b) = exp(-G_{\Omega_b}.LAD.l_b) \tag{6}$$

where G_{Ω_b} is a coefficient characterizing the spatial distribution of leaf orientations in the crown volume, LAD is the Leaf Area Density in the volume and l_b the length of the beam path in the crown volume. Assuming the B beams constitute a regular and dense sampling of the whole volume, the approximated directional STAR of the turbid volume, $\widehat{STAR}_\Omega$, can then be computed as [87]:

$$\widehat{STAR}_\Omega = \sum_{b=1}^{B} S_b(1 - p_0(b))/S \tag{7}$$

where S_b is the cross section area of a beam. This model-based definition of the STAR can be compared to STAR from Eq. 5 to evaluate the quality of light model assumptions. The resulting difference characterizes the error due to the model's underlying hypotheses (homogeneity/randomness of the foliage distribution, negligibility of leaf size, ...) with respect to the actual canopies [86].

In `PlantGL`, both STAR quantities, *i.e.* the projection-based and turbid-medium-based STARs, can be computed from a plant mockup using the high-level library functions. The projection based STAR of a given virtual canopy can be computed by counting the number of vegetation pixels in a virtual picture obtained by projecting virtual plant canopies using an orthographic camera [86] and multiplying by the size of a pixel. This would be expressed as follows in `PlantGL`:

```
def star(leaves,dir):
  Viewer.display(leaves)
  Viewer.camera.setOrthographic()
  Viewer.camera.setDirection(dir)
  proj, nbpixel,
   pixelsize = Viewer.frameGL.getProjectionSize()
  return proj/ surface(leaves)
```

For the turbid medium based STAR, the envelope of the tree crown must first be computed. Then, a set of beams of direction Ω are cast and their interceptions and resulting length in the crown volume are computed. A sketch of such a code would be as follows:

```
def star (leaves, g, dir, up, right, beam_radius) :
  hull = fit ('convexhull', leaves)
  lad = surface (leaves) / volume(hull)
  bbx = BoundingBox (hull, dir, up, right)
  Viewer.display (hull)
  pos = bbx.upperRightCorner ()
  interception = 0
  for rshift in range (bbx.size ().y/beam_radius) :
    for upshift in range (bbx.size ().z/beam_radius) :
     intersections = Viewer.castRay (pos-rshift*right-
  upshift*up, dir)
    p0 = 1
    for intersection in intersections:
     length = norm (intersection.out-intersection.in)
    p0 *= exp(-g*lad*length)
  interception += (1-p0)* (beam_radius**2)
return interception / surface (leaves)
```

with `intersection` containing the in and out intersection points of the ray and a hull and `intersections` being a list of such structure. The STAR with ray casting took 0.3 s for each direction to be computed with `PlantGL` and the approximated one took 10 s using 50000 rays.

4.3.3. Simulation at community scale

Detailed plant models, at the level of branches and leaves, do not always correspond to the most adequate level for expressing knowledge in plant models. `PlantGL` provides a number of ways to deal with abstract representation of plants at different scales. In particular, the various envelope models defined in Section 3.2.1 can be used as abstract means to model plant crown bulk. Such models are useful for instance in the modeling of plant communities, where competition for space has been shown to be a key structuring factor [88].

In the following example, we illustrate how natural scenes containing thousands of plants distributed in a realistic manner can be built with `PlantGL`, taking into account competition for space. It is inspired by [31] which is an extension of [89,90] to the use of more complex crown shapes.

The ecosystem synthesis starts with the generation of a set of coarse individuals with height, crown radius and crown base height determined from density and allometric functions.

Individuals are *fagus* beech trees with different classes of ages. Allometric functions of the *Fagacées* model [91] are used to determine the heights and radius values as a function of tree age. The spatial distribution of these plants is generated using a stochastic point process. For this, we use a Gibbs process [92,93] defined as a pairwise interaction function $f(p_i,p_j)$, that represents the cost associated with the presence of two given plants at positions p_i and p_j, respectively. Positive cost values will lead to repulsion between trees while negative ones lead to attraction. A realization of this process is intended to minimize the global cost $F = \sum_{i \neq j} f(p_i, p_j)$, defined as the sum of the costs associated with each pair of points. The Gibbs process is simulated with a classical depletion-replacement iterative algorithm [94].

Classically, the cost function is used to model neighbor competition and is defined as a function of the crown radii and positions of the trees. The cost function of two trees i and j, characterized by shapes with constant radius, may be chosen for instance proportional to the difference between the sum of the crown radii and the distance between p_i and p_j. For asymmetric shapes, the same function can be used where radii of trees now correspond to the radius of each envelope in the direction defined by the tree positions p_i and p_j. In addition, both the position and the different parameters of the crown envelope can now be changed in the depletion-replacement algorithm. Fig. 19 illustrates the 3D output of such a process.

From this set of coarse individuals, detailed plant representations can be inferred and assembled into a complete scene. For this, different generation methods either available in `PlantGL` or outside of the software can be used. In our example, we generated the beech trees with the L-systems models using the generative procedure described in [30].

Bushes and flowers were generated using `PlantGL` and `Python` as presented in Section 3.3 and added to the scene. Finally, a digitized walnut tree [65] was also added to illustrate how scenes may be created in `PlantGL` using a range of classical date sources.

The final rendering was made with `Povray` [68]. Each plant geometric model was converted and assembled in

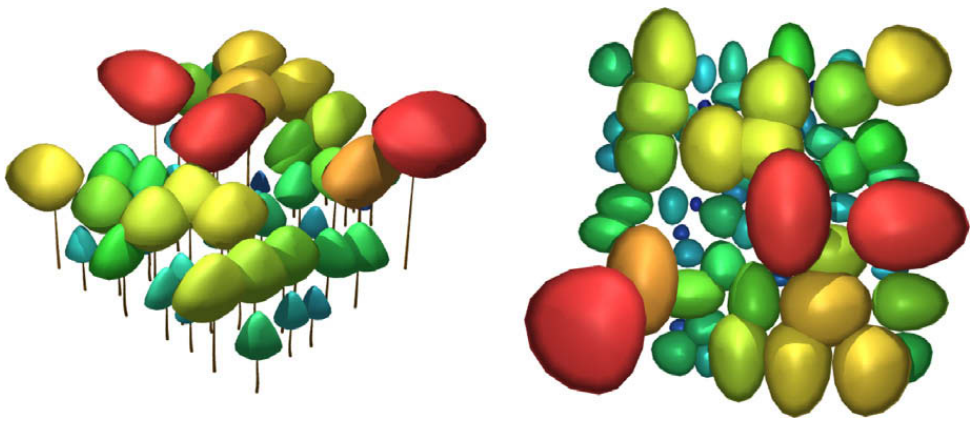

Fig. 19. A front and top view of a generated stand at the crown scale. Different colors are used to differentiate various layers of vegetation.

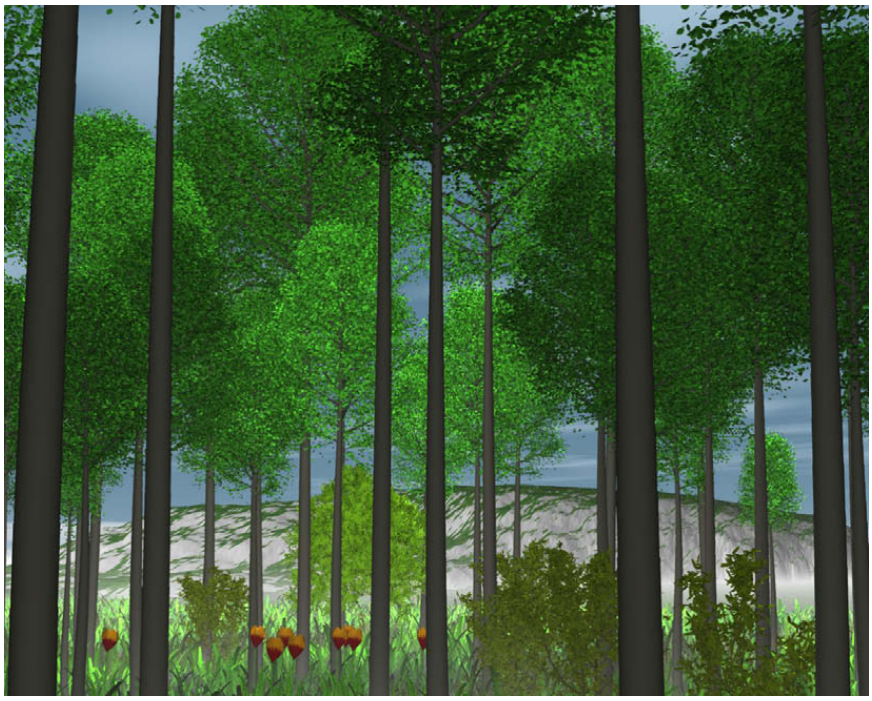

Fig. 20. A community of plants generated from a Gibbs process [31]. The scene is made of plants from different sources: beech trees of different sizes and ages where generated using the ecosystem model presented in Section 4.3.3. A walnut tree corresponding to a 3D digitizing of a real plant built using AMAPmod/VPlants, virtual bushes flowers and grass created procedurally in `Python` with `PlantGL`.

this format. Fig. 20 illustrates the resulting scene. The computation of the distribution of the 56 trees on the terrain using the depletion-replacement algorithm with 4000 iterations is made using pure `Python` and took 5 min. Generation of individual tree structure using L-systems took 20 s per tree. The final `Povray` rendering took 5 min.

5. Conclusion

In this paper, we presented a new open-software library for the geometric modeling of plants built on the top of the `Python` programming language. The library provides a set of geometric models that are useful to represent various types of plant structures at different scales, ranging from tissues to plant communities. In particular, it contains original geometric components such as dedicated parametric envelopes for crown shape representation and tissues for representing plants at cell scale in 2D or 3D. Branching systems can be created either procedurally or by importing them from plant growth simulation platforms, such as `LStudio/VLab`. The resulting plant geometric models can be easily analysed using `Python` and `PlantGL` high level algorithms. The different features of the `PlantGL` library have been illustrated on applications involving plants at different scales and showing its use at various stages of a modeling process.

Acknowledgments

The authors thank P. Prusinkiewicz for kindly making the `LStudio/VLab` software available to them, D. Da Silva for STAR computation on the apple tree and the shadow images, P. Barbier de Reuille and Y. Caraglio for their contribution to some images, and H. Sinoquet, E. Costes, F. Danjon, C.-E. Parveau and J. Traas for making 3D digitized plants or tissues available to them. This work has been partially supported by ANR projects *NatSim* and *Virtual Carpel*.

Appendix A. Implementation issues

Different issues have to be addressed in order to implement an efficient scene-graph that preserves performance and flexibility. Literature on scene-graphs [95] offers various interesting solutions that were formalized as design patterns and which inspired our implementation.

Memory management: Scene-graphs have to deal with a large number of geometric elements. This is particularly true for natural scenes. Memory consumption is thus an issue. Repetitive structures such as trees enable massive use of *instantiation*. This technique, however, implies that the same object is referenced several times. Allocation and deallocation of this object in memory can thus be problematic. To address this issue, we use *Reference counting pointers* [96] which manages pointers to dynamically allocated objects. They are responsible for automatic deletion of the objects when no longer needed.

Visitor actions: As stated in Section 2.3.2, algorithm polymorphism is implemented using the visitor design pattern to avoid modification of object interfaces. With such an approach, algorithms can be added without any modification of the hierarchy of objects. However, at runtime, matching a particular data structure to its appropriate algorithms is not

a trivial task. For this we need to determine at run time the actual type of a node before choosing the algorithm. To do that, each class of the hierarchy implement an *apply* method that takes an action object as an argument. The method makes a call to the action, passing the node as an argument, and the action executes the appropriate algorithm depending on the actual node type.

Wrappers: The seamless transition between `C++` and `Python` is ensured by the `Boost.Python` library [38]. In contrast with concurrent tools, the interaction between `C++` and `Python` is encoded explicitly in `C++`. This wrapping code is then compiled into a dynamic library, usable as a `Python` module. On one hand, `Boost.Python` maps `C++` classes and their interfaces to corresponding `Python` classes. On the other hand, it transparently converts `Python` objects back to `C++` pointers or references, thus providing dynamic, run-time dependent interaction between `C++` based objects. Since `Boost.Python` is based on advanced meta-programming techniques, the code wrapping mainly consists of simple declaration of entry points in the library and is automatically translated into conversion functions.

Graphic performances: Plant geometrical descriptions generally rely on large number of triangles. In order to minimize time cost for sending all the triangles to the GPU, the triangle points are packed into arrays that can be sent in one command to the card. Moreover, `OpenGL` commands for drawing shapes instantiated multiple times are packed into display lists that can be reused efficiently when needed.

Multi-threading: Viewer and shell processing are done in separate threads. For this, the `Qt` threads implementation is used. Inter-thread communication is made using `Qt` event dispatch mechanism which is extended with synchronized dispatch using mutual exclusion lock (also called mutex) [97].

Appendix B. Mathematical details of envelope models

B.1. Asymmetric hull

This envelope model is defined by six control points and two shape factors C_T and C_B. The four points P_1 to P_4 define the peripheral line L. For x and y, points of L form four elliptical quarters centered on the origin. For z, their height is defined as an interpolation of the heights of the control points. To be continuous at the control points, we used factors $\cos^2\theta$ and $\sin^2\theta$ in the interpolation. Thus, a point $P_{\theta,i,j}$ of a quarter of L between control points P_i and P_j with i, $j \in [(1,2),(2,3),(3,4),(4,1)]$, located at an angle $\theta \in [0, \frac{\pi}{2})$, is defined as

$$P_{\theta,i,j} = \left[r_{P_i}\cos\theta, r_{P_j}\sin\theta, z_{P_i}\cos^2\theta + z_{P_j}\sin^2\theta\right]. \tag{B.1}$$

Points of L are connected to the top and base points with quarters of super-ellipses of degrees C_T and C_B, respectively. Letting $P_l \in L$ and P_T be the top point of the envelope, the super-ellipse quarter connecting P_l and P_T is defined as

$$\left\{P = (\theta, r, z) \,|\, \frac{(r - r_{p_T})^{C_T}}{(r_{P_l} - r_{p_T})^{C_T}} + \frac{(z - z_{p_T})^{C_T}}{(z_{P_l} - z_{p_T})^{C_T}} = 1\right\}. \tag{B.2}$$

An equivalent equation is obtained for super-ellipse quarters using P_B and C_B instead of P_T and C_T.

B.2. Extruded Hull

The extruded envelope is defined from a horizontal profile H and a vertical profile V. First, two fixed points, B and T, at the top and bottom of the vertical profile, respectively, are defined (see Fig. 6b). V is split into two open profile curves V_l and V_r, the left and the right part of V, respectively, with their first and last points equal to B and T. A slice $S(u)$ is thus defined using two mapping points $V_l(u)$ and $V_r(u)$. Its span vector $\overrightarrow{S(u)}$ is set to $\overrightarrow{V_l(u)V_r(u)}$.

On the horizontal profile H, two anchor points, H_l and H_r, are defined. A horizontal section of the extruded hull is computed at $S(u)$ so that the resulting curve fits inside V (see Fig. 6b and c). Finally, to map $\overrightarrow{H_lH_r}$ to $\overrightarrow{S(u)}$, the transformation is a composition of a *translation* from H_l to $V_l(u)$, a *rotation* around the $\vec{y}$ axis of an angle $\alpha(u)$ equal to the angle between the $\vec{x}$ axis and $\overrightarrow{S(u)}$, and a *scaling* by the factor $\|\overrightarrow{S(u)}\|/\|\overrightarrow{H_lH_r}\|$.

The equation of the Extruded Hull surface is thus:

$$S(u,v) = V_l(u) + \frac{\|\overrightarrow{S(u)}\|}{\|\overrightarrow{H_lH_r}\|} * R_y(\alpha(u))(H(v) - H_l) \tag{B.3}$$

B.3. Skinned Hull

The envelope of a skinned hull is a closed skinned surface which interpolates a set of profiles $\{P_k(u), k = 0,\ldots,K\}$ positioned at angle $\{\alpha_k, k = 0,\ldots,K\}$ around the z axis. Similarly to an extruded hull, all vertical profiles are split into two open profile curves homogeneously parameterized. We assume thus that all these profiles $P_k(u)$ are non-rational B-spline curves with common degree p and number n of control points $P_{i,k}$ (see [52] for homogenization details). From these profiles, a variational profile Q can be defined that gives a section for each angle α around the rotation axis (see Fig. 7b). It is defined as

$$Q(u,\alpha) = \sum_{i=0}^{n} N_{i,p}(u) Q_i(\alpha) \tag{B.4}$$

where the $N_{i,p}(u)$ are the pth-degree B-spline basis functions and the $Q_i(\alpha)$ are a variational form of control points. Q interpolates all the profiles P_k. Therefore, $Q_i(\alpha)$ are computed using a global interpolation method [52] on the control points $P_{i,k}$ at α_k with $k \in [0,K]$ For this, let q be the chosen degree of the interpolation such as $q < K$. $Q_i(\alpha)$ are defined as

$$Q_i(\alpha) = \sum_{j=0}^{K} N_{j,q}(\alpha) R_{i,j} \tag{B.5}$$

where the control points $R_{i,j}$ are computed by solving interpolation constraints that results in a system of linear equations:

$$\forall i \in [0,n], \quad \forall k \in [0,K], \quad P_{i,k} = Q_i(\alpha_k) = \sum_{j=0}^{K} N_{j,q}(\alpha_k) R_{i,j} \tag{B.6}$$

Geometrically, the surface of the skinned hull is obtained by rotating $Q(u,\alpha)$ about the z axis, α being the rotation angle. It is thus defined as

$$S(u,\alpha) = (\cos\alpha Q_x(u,\alpha), \sin\alpha Q_x(u,\alpha), Q_y(u,\alpha)). \quad \text{(B.7)}$$

References

[1] P. Prusinkiewicz, A. Lindenmayer, J. Hanan, Development models of herbaceous plants for computer imagery purposes, in: SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, ACM, New York, NY, USA, 1988, pp. 141–150.

[2] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, C. Puech, Plant models faithful to botanical structure and development, in: SIGGRAPH'88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, New York, NY, USA, 1988, pp. 151–158.

[3] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants, Springer-Verlag, New York, 1990.

[4] J. Weber, J. Penn, Creation and rendering of realistic trees, in: SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, NY, USA, 1995, pp. 119–128.

[5] B. Lintermann, O. Deussen, Interactive modeling of plants, IEEE Computer Graphics and Applications 19 (1) (1999) 56–65.

[6] P. Prusinkiewicz, L. Mündermann, R. Karwowski, B. Lane, The use of positional information in the modeling of plants, in: SIGGRAPH'01, Computer Graphics, ACM, Los Angeles, California, 2001, pp. 36–47.

[7] I. Shlyakhter, M. Rozenoer, J. Dorsey, S. Teller, Reconstructing 3d tree models from instrumented photographs, IEEE Computer Graphics and Applications 21 (3) (2001) 53–61.

[8] B. Neubert, T. Franken, O. Deussen, Approximate image-based tree-modeling using particle flows, ACM Transactions on Graphics (Proc. of SIGGRAPH) 26 (3) (2007).

[9] P. Tan, G. Zeng, J. Wang, S.-B. Kang, L. Quan, Image-based tree modeling, ACM Transactions on Graphics (Proc. of SIGGRAPH 2007) 26 (3) (2007).

[10] A. Cescatti, Modelling the radiative transfer in discontinuous canopies of asymmetric crown. I. model structure and algorithms, Ecological Modelling 101 (1997) 263–274.

[11] H. Sinoquet, P. Rivet, C. Godin, Assessment of the three-dimensional architecture of walnut trees using digitising, Silva Fennica 31 (3) (1997) 265–273.

[12] C. Godin, Representing and encoding plant architecture: a review, Annals of Forest Science 57 (05-juin) (2000) 413–438.

[13] F. Danjon, H. Sinoquet, C. Godin, F. Colin, M. Drexhage, Characterisation of structural tree root architecture using 3d digitising and amapmod software, Plant and Soil 211 (2) (1999) 241–258.

[14] J.B. Evers, J. Vos, C. Fournier, B. Andrieu, M. Chelle, P.C. Struik, Towards a generic architectural model of tillering in gramineae as exemplified by spring wheat (triticum aestivum), New Phytologist 166 (3) (2005) 801–812.

[15] F. Boudon, C. Godin, C. Pradal, O. Puech, H. Sinoquet, Estimating the fractal dimension of plants using the two-surface method. An analysis based on 3d-digitized tree foliage, Fractals 14 (3) (2006) 149–163.

[16] R.-S. Smith, C. Kuhlemeier, P. Prusinkiewicz, Inhibition fields for phyllotactic pattern formation: a simulation study, Canadian Journal of Botany 84 (2006) 1635–1649.

[17] A. Lindenmayer, Mathematical models for cellular interactions in development, I & II, Journal of Theoretical Biology (1968) 280–315.

[18] R. Karwowski, P. Prusinkiewicz, Design and implementation of the L+C modeling language, Electronic Notes in Theoretical Computer Science 86.

[19] P. Prusinkiewicz, R. Karwowski, B. Lane, The L+C plant modeling language, in: J. Vos et al. (Eds.), Functional–Structural Plant Modelling in Crop Production, Springer, 2007.

[20] W. Kurth, Growth Grammar Interpreter GROGRA 2.4: a software tool for the 3-dimentional interpretation of stochastic, sensitive growth grammar in the context of plant modelling, Introduction and reference manual, Forschungszentrum Waldokosysteme der Universitat Gottingen, 1994.

[21] O. Kniemeyer, G.-H. Buck-Sorlin, W. Kurth, A graph grammar approach to artificial life, Artificial Life 10 (4) (2004) 413–431.

[22] G.-H. Buck-Sorlin, O. Kniemeyer, W. Kurth, Barley morphology genetics and hormonal regulation of internode elongation modelled by a relational growth grammar, New Phytologist 10 (4) (2005) 413–431.

[23] J.-F. Barczi, P. de Reffye, Y. Caraglio, Essai sur l'identifcation et la mise en oeuvre des paramétres nécessaires àla simulation d'une architecture végétale: le logiciel amapsim., in: J. Bouchon, P. de Reffye, D. Barthélémy (Eds.), Modélisation et Simulation de l'Architecture des Végétaux, INRA Editions, 1997, pp. 205–254.

[24] O. Deussen, B. Lintermann, Digital Design of Nature. Computer Generated Plants and Organics, Springer-Verlag, 2005.

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

[26] C. Godin, E. Costes, H. Sinoquet, A method for describing plant architecture which integrates topology and geometry, Annals of Botany 84 (1999) 343–357.

[27] E. Costes, H. Sinoquet, J.-J. Kelner, C. Godin, Exploring within-tree architectural development of two apple tree cultivars over 6 years, Annals of Botany 91 (2003) 91–104.

[28] P. Ferraro, C. Godin, P. Prusinkiewicz, Toward a quantification of self-similarity in plants, Fractals 13 (2) (2005) 91–109.

[29] Y. Guédon, Y. Caraglio, P. Heuret, E. Lebarbier, C. Meredieu, Analyzing growth components in trees, Journal of Theoretical Biology 248 (2007) 418–447.

[30] F. Boudon, P. Prusinkiewicz, C. Federl, P. Godin, R. Karwowski, Interactive design of bonsai tree models, Computer Graphics Forum (Proc. of Eurographics'03) 22 (3) (2003) 591.

[31] F. Boudon, G. Le Moguedec, Déformation asymétrique de houppiers pour la génération de représentations paysagères réalistes, Revue Electronique Francophone d'Informatique Graphique (REFIG) 1 (1) (2007) 9–19.

[32] F. Danjon, T. Fourcaud, D. Bert, Root architecture and wind-firmness of mature pinus pinaster, New Phytologist 168 (2) (2005) 387–400.

[33] P. Barbier de Reuille, I. Bohn-Courseau, C. Godin, J. Traas, A protocol to analyse cellular dynamics during plant development, The Plant Journal 44 (2005) 1045–1053.

[34] D. Da Silva, F. Boudon, C. Godin, O. Puech, C. Smith, H. Sinoquet, A critical appraisal of the box counting method to assess the fractal dimension of tree crowns, Lecture Notes in Computer Sciences (Proceedings of the 2nd International Symposium on Visual Computing) 4291 (2006) 750–751.

[35] G. Louarn, Y. Guédon, J. Lecoeur, E. Lebon, Quantitative analysis of the phenotypic variability of shoot architecture in two grapevine cultivars (vitis vinifera l.), Annals of Botany 99 (3) (2007) 425–437.

[36] J. Chopard, C. Godin, J. Traas, 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 Cell And Tissues, Oxford, UK, 2007.

[37] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[38] Boost C++ libraries (1998–2006). Available from: <http://www.boost.org/>.

[39] P.S. Strauss, R. Carey, An object-oriented 3d graphics toolkit, in: SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, ACM Press, New York, NY, USA, 1992, pp. 341–349.

[40] E. Welzl, Smallest enclosing disks (balls and ellipses), New Results and New Trends in Computer Science 555 (1991) 359–370.

[41] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Transactions on Mathematical Software 22 (4) (1996) 469–483.

[42] N. Greene, Voxel space automata: modeling with stochastic growth processing in voxel space, Computer Graphics 23 (3) (1989) 175–184.

[43] F. Boudon, C. Nouguier, C. Godin, GEOM module manual. I. User's guide, Document de travail du Programme Modélisation des plantes 3-2001, CIRAD, 2001.

[44] R. Carey, G. Bell, The Annotated VRML 2.0 Reference Manual, Addison-Wesley, 2002.

[45] H. Vogel, A better way to construct the sunflower head, Mathematical Biosciences 44 (1979) 179–189.

[46] J. Phattaralerphong, H. Sinoquet, A method for 3d reconstruction of tree crown volume from photographs: assessment with 3d-digitized plants, Tree Physiology 25 (2005) 122942.

[47] A. Reche-Martinez, I. Martin, G. Drettakis, Volumetric reconstruction and interactive rendering of trees from photographs, ACM Transactions on Graphics 23 (3) (2004) 720–727.

[48] H. Horn, The Adaptive Geometry of Trees, Princeton University Press, Princeton, NJ, 1971.

[49] H. Koop, Silvi-star: A comprehensive monitoring system, Forest Dynamics (1989) 229.
[50] P. Birnbaum, Modalités d'occupation de l'espace par les arbres en forêts guyanaise, Master's thesis, Université Paris VI, 1997.
[51] C.D. Woodward, Skinning techniques for interactive b-spline surface interpolation, Computer Aided Design 20 (10) (1988) 441–451.
[52] L.A. Piegl, W. Tiller, The Nurbs Book, second ed., Springer, 1997.
[53] L.A. Piegl, W. Tiller, Surface skinning revisited, The Visual Computer 18 (4) (2002) 273–283.
[54] M. Barnsley, Fractals Everywhere, Academic Press, Boston, 1988.
[55] K. Falconer, Techniques in Fractal Geometry, John Wiley and Sons, 1997.
[56] C. Godin, Y. Caraglio, A multiscale model of plant topological structures, Journal of Theoretical Biology 191 (1998) 1–46.
[57] R. Mech, M. James, M. Hammel, J. Hanan, P. Prusinkiewicz, CPFG v4.0 user manual, The University of Calgary, 2005.
[58] F. Boudon, Représentation géométrique multi-échelles de l'architecture des plantes, Ph.D. thesis, Université de Montpellier II, 2004.
[59] P. Prusinkiewicz, Graphical applications of L-systems, in: Proceedings on Graphics Interface'86/Vision Interface'86, Canadian Information Processing Society, Toronto, Ont., Canada, 1986, pp. 247–253.
[60] C. Godin, Y. Guédon, E. Costes, Exploration of a plant architecture databases with the AMAPmod software illustrated on an apple-tree hybrid family, Agronomie 19 (3–4) (1999) 163–184.
[61] B. Adam, N. Dones, H. Sinoquet, Vegestar v.3.1. a software to compute light interception and photosynthesis by 3d plant mock-ups, in: C. Godin (Ed.), Fourth International Workshop on Functional-Structural Plant Models, Montpellier, France, 2004, p. 414.
[62] E.H. Spanier, Algebraic Topology, McGraw-Hill, New York, 1966.
[63] Cgal, Computational Geometry Algorithms Library. Available from: <http://www.cgal.org>.
[64] H. Sinoquet, B. Andrieu, The geometrical structure of plant canopies: characterization and direct measurements methods, in: C. Varlet-Grancher, R. Bonhomme, H. Sinoquet (Eds.), Crop structure and light microclimate, Vol. 0, INRA Editions, Paris, 1993, pp. 131–158.
[65] H. Sinoquet, P. Rivet, C. Godin, Assessment of the three-dimensional architecture of walnut trees using digitizing, Silva Fennica 3 (1997) 265–273.
[66] C.-E. Parveaud, Propriétés radiatives des couronnes de noyers (Juglans nigra × J. regia) et croissance des pousses annuelles—influence de la géométrie du feuillage, de la position des pousses et de leur climat radiatif, Ph.D. Thesis, Université de Montpellier II, France, 2006.
[67] E. Casella, H. Sinoquet, A method for describing the canopy architecture of coppice poplar with allometric relationships, Tree Physiology 23 (2003) 1153–1170.
[68] T.P.-R. Team, Persistence of vision raytracer (1991–2006). Available from: <http://www.povray.org/>.
[69] B.B. Mandelbrot, The Fractal Geometry of Nature, W.N. Freeman, New York, USA, 1983.
[70] E. Costes, H. Sinoquet, C. Godin, J.J. Kelner, 3D digitizing based on tree topology: application to study the variability of apple quality within the canopy, Acta Horticulturae 499 (1999) 271–280.
[71] M. Barnsley, S. Demko, Iterated function systems and the global construction of fractals, Royal Society of London Proceedings Series A 399 (1985) 243–275.
[72] A.L. Oppelt, W. Kurth, H. Dzierzon, G. Jentschke, D.L. Godbold, Structure and fractal dimensions of root systems of four co-occurring fruit tree species from Botswana, Annals of Forest Science 57 (2000) 463–475.
[73] J. Runions, T. Brach, S. Kuhner, Photoactivation of GFP reveals protein dynamics within the endoplasmic reticulum membrane, Journal of Experimental Botany 57 (1) (2006) 43–50.
[74] F.G. Feugier, Models of vascular pattern formation in leaves, Ph.D. thesis, Pierre et Marie Curie, 2005.
[75] A.-G. Rolland-Lagan, E. Coen, S.J. Impey, J.A. Bangham, A computational method for inferring growth parameters and shape changes during development base clonal analysis, Journal of Theoretical Biology 232 (2005) 157–177.
[76] H. Jönsson, M. Heisler, G.V. Reddy, V. Agrawal, V. Gor, B.E. Shapiro, E. Mjolsness, E.M. Meyerowitz, Modeling the organization of the WUSCHEL expression domain in the shoot apical meristem, Bioinformatics 21 (Suppl. 1) (2005) i232–i240, doi:10.1093/bioinformatics/bti1036.
[77] H. Jönsson, M.G. Hesler, B.E. Shapiro, E.M. Meyerowitz, E. Mjolsness, An auxin-driven polarized transport model for phyllotaxis, PNAS 103 (5) (2006) 1633–1638.
[78] P. Barbier de Reuille, I. Bohn-Courseau, K. Ljung, H. Morin, N. Carraro, C. Godin, J. Traas, Computer simulations reveal properties of the cell–cell signaling network at the shoot apex in Arabidopsis, PNAS 103 (5) (2006) 1627–1632.
[79] C. Smith, On vertex-vertex systems and their use in geometric and biological modelling, Ph.D. Thesis, University of Calgary, 2006.
[80] M. Heisler, H. Jönsson, Modelling meristem development in plants, Current Opinion in Plant Biology 10 (2007) 92–97.
[81] A.-G. Rolland-Lagan, J.A. Bangham, E. Coen, Growth dynamics underlying petal shape and asymmetry, Nature 422 (13) (2003) 161–163.
[82] J. Nakielski, Tensorial model for growth and cell division in the shoot apex, in: A. Carbone, M. Gromov, P. Prusinkiewicz (Eds.), Pattern Formation in Biology, Vision and Dynamics, World Scientific, Singapore, 2000, pp. 252–267.
[83] P. Oker-Blom, H. Smolander, The ratio of shoot silhouette area to total needle area in scots pine, Forest Science 34 (1988) 894–906.
[84] J.A. Den Dulk, The interpretation of remote sensing, a feasibility study, Ph.D. Thesis, Wageningen university, 1989.
[85] P. Moon, D. Spencer, Illumination from a non-uniform sky, Transactions of the Illumination Engineering Society 37 (1942) 707712.
[86] H. Sinoquet, G. Sonohat, J. Phattaralerphong, C. Godin, Foliage randomness and light interception in 3-d digitized trees: an analysis from multiscale discretization of the canopy, Plant, Cell and Environment 28 (9) (2005) 1158–1170.
[87] H. Sinoquet, C. Varlet-Granchet, R. Bonhomme, Modelling radiative transfer within homogeneous canopies: basic concepts, in: C. Varlet-Grancher, R. Bonhomme, H. Sinoquet (Eds.), Crop structure and light microclimate, INRA Editions, Paris, 1993, pp. 131–158.
[88] A. Franc, S. Gourlet-Fleury, N. Picard, Une introduction à la modélisation des forêts hétérogènes, ENGREF, Nancy, 2000.
[89] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, P. Prusinkiewicz, Realistic modeling and rendering of plant ecosystems, Computer Graphics 32 (Annual Conference Series) (1998) 275–286.
[90] B. Lane, P. Prusinkiewicz, Generating spatial distributions for multilevel models of plant communities, in: Proceedings of Graphics Interface 2002, Calgary, Alberta, 2002, pp. 69–80.
[91] G. Le Moguédec, J. Dhôte, Présentation du modèle Fagacées, Tech. rep., LERFOB, INRA, Nancy, France, 2002.
[92] P. Diggle, Statistical Analysis of Spatial Point Patterns, Academic Press, London, UK, 1983.
[93] F. Goreaud, Apport de l'analyse de la structure spatiale en forêt tempérée à l'étude de la modélisation des peuplements complexes, Ph.D. Thesis, ENGREF, 2004.
[94] B. Rippley, Simulating spatial patterns: dependent samples from a multivariate density, Applied Statistic 28 (1979) 109–112.
[95] H. Sowizral, Scene graphs in the new millennium, IEEE Computer Graphics and Applications 20 (1) (2000) 56–57.
[96] S. Meyers, More Effective C++: 35 New Ways to Improve Your Programs and Designs, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
[97] M. Raynal, D. Beeson, Algorithms for Mutual Exclusion, MIT Press, Cambridge, MA, USA, 1986.

Chapitre 4

Modèle de calcul d'ordre supérieur pour coupler analyse et simulation dans les workflows scientifiques

La programmation visuelle permet de représenter de façon explicite la structure fonctionnelle d'un programme et offre la possibilité à un modélisateur de l'analyser et de la modifier de façon interactive. Dans un chapitre précédent, nous avons montré, à travers *OpenAlea* et l'environnement de programmation visuelle *VisuAlea*, comment cela favorisait la réutilisation de composants existants et les approches modulaires.

Cependant, nous avons aussi constaté que le modèle de dataflow induit de fortes limitations pour la modélisation, car il ne permet pas de représenter des cycles et des structures de contrôle nécessaires à la simulation. En effet, l'impossibilité de boucler sur la structure ne permet pas de modéliser un système dynamique. Des simulateurs peuvent être intégrés dans la plateforme OpenAlea, mais cette limitation empêche la réalisation d'un modèle structure-fonction complêtement modulaire.

Ce chapitre décrit l'extension du modèle de dataflow en utilisant le λ-calcul afin de pouvoir exprimer des dataflows d'ordre supérieur. Au lieu de passer en argumant des valeurs aux tâches du dataflow, ce modèle permet de passer en argument d'opérateurs algébriques un dataflow calculé à la volée. On peut donc, avec les opérateurs appropriés, facilement représenter des boucles ou tout autre structure de contrôle et cela sans recourir à des cycles dans la structure du dataflow. Pratiquement, cela permet de pouvoir modéliser la rétro-action entre la structure et la fonction d'une plante, et non plus seulement le chaînage de modèles, tout en utilisant des dataflows d'ordre supérieurs et des opérateurs algébriques.

Ce chapitre se divise en plusieurs sections. Tout d'abord nous présentons ce qu'est un modèle de calcul et notamment les modèles de calcul dataflow et ceux à évenements

discret. La deuxième section porte sur la description détaillée des principaux concepts utilisés et des choix réalisés pour la réalisation du modèle de dataflow.

La troisième section présente les différentes stratégies d'exécution du graphe de composants et décrit les algorithmes d'évaluation. Certaines fonctionnalités avancées y sont présentées telle que la possibilité d'exprimer des boucles, de coupler un modèle de dataflow avec un modèle simple à événement discret ainsi que la parallélisation. Finalement, le modèle de programmation d'OpenAlea sera discuté dans la dernière section.

La contribution de ce chapitre à cette thèse est de montrer comment il a été possible, en utilisant l'abstraction offerte par le modèle de calcul, d'étendre les avantages offerts par les workflows scientifiques au domaine de l'analyse et à la simulation de systèmes complexes.

L'impact de ce travail peut être quantifié par le nombre de citations (21 selon google scholar), ainsi que par le fait qu'il soit à la base d'autres travaux comme InfraPhenoGrid, une infrastructure pour le phénotypage haut-débit sur la grille (Pradal et al., 2017 ; Pradal et al., 2018).

Ce chapitre est la version originale du papier de conférence :

OpenAlea : Scientific Workflows Combining Data Analysis and Simulation C. Pradal, C. Fournier, P. Valduriez et S. Cohen-Boulakia.
Publié dans ACM Proceedings of the 27th International Conference on Scientific and Statistical Database Management, 2015 Vol 11 :1–6

OpenAlea: Scientific Workflows Combining Data Analysis and Simulation

Christophe Pradal
UMR AGAP, CIRAD and Inria
Montpellier, France
christophe.pradal@cirad.fr

Christian Fournier
INRA
Montpellier, France
christian.fournier@inra.fr

Patrick Valduriez
Inria and LIRMM, Montpellier, France
Patrick.Valduriez@inria.fr

Sarah Cohen-Boulakia
Inria, Montpellier, France
LRI CNRS 8623, U.Paris Sud
cohen@lri.fr

ABSTRACT

Analyzing biological data (e.g., annotating genomes, assembling NGS data...) may involve very complex and interlinked steps where several tools are combined together. Scientific workflow systems have reached a level of maturity that makes them able to support the design and execution of such in-silico experiments, and thus making them increasingly popular in the bioinformatics community.

However, in some emerging application domains such as system biology, developmental biology or ecology, the need for data analysis is combined with the need to model complex multi-scale biological systems, possibly involving multiple simulation steps. This requires the scientific workflow to deal with retro-action to understand and predict the relationships between structure and function of these complex systems. OpenAlea (openalea.gforge.inria.fr) is the only scientific workflow system able to uniformly address the problem, which made it successful in the scientific community. One of its main originality is to introduce *higher-order dataflows* as a means to uniformly combine classical data analysis with modeling and simulation.

In this demonstration paper, we provide for the first time the description of the OpenAlea system involving an original combination of features. We illustrate the demonstration on a high-throughput workflow in phenotyping, phenomics, and environmental control designed to study the interplay between plant architecture and climatic change.

1. INTRODUCTION

Classical bioinformatics analysis (e.g. annotating genomes, building phylogenetic trees, assembling NGS data) involves the management and processing of huge data sets together with the chaining of numerous complex and interlinked tools. Scientific workflow systems aim at facilitating and rationalizing the design and management of such tasks. They clearly separate the workflow specification from its execution and offer useful capabilities on both aspects. Among others, they may provide a user interface to design workflows by composing tools [15], a scheduler to optimize the processing of huge amounts of data [6], a provenance module [4] to keep track of the data used and generated during an execution and ensure the reproducibility of the experiments [18].

However, the complexity of biological analysis increases in emergent interdisciplinary domains. This is especially the case in domains addressing the study of complex multi-scale systems that require numerical simulations. In system biology for instance, analyzing the emergent behavior of a large number of interactions within a biological system requires simulating the interplay between the topological and geometrical development of the structure and its biological functioning. This involves coupling models from different disciplines, integrating experimental data from various sources at different scales (gene, cell, tissue, organism and population), and analyzing the reconstructed system with numerical experiments.

While scientific workflow systems have mainly been designed to support data analysis and visualization [15, 8, 9, 1], only a few systems have attempted to support iteration or simulation [1, 15]. Most of the systems use either control flow edges or define loops in the workflow specification with specific routing nodes (e.g. switch [7]). Kepler [1] uses black box actors with different models of computation to provide iteration processes. These solutions can lead designing overly complex workflows that are difficult to understand, reuse, and maintain [1]. Expressing control flow (iteration) in scientific workflows is actually a difficult problem due to the absence of state variable and side-effect.

To address this problem, we have introduced the concept of λ-dataflow, which is inspired from the λ-calculus used in Functional Programming.

λ-dataflow makes use of higher-order constructs in the context of dataflows theory and thus allows to represent control flow using algebraic operators [6] (e.g., conditionals, map/reduce...). λ-dataflow allows to model retro-action. The OpenAlea system [17] that we introduce in this paper is

 ISBN 978-1-4503-3709-0/15/06 ...$15.00
DOI: http://dx.doi.org/10.1145/2791347.2791365.

a workflow system based on λ-dataflow which is able to uniformly deal with classical data analysis, visualization, modeling and simulation tasks.

In this paper, we show how the notion of λ-dataflow allows OpenAlea to uniformly deal with workflows involving data analysis and simulation steps. Our demonstration introduces the capabilities of OpenAlea on a workflow involving high-throughput phenotyping, phenomics, and environmental control, to study the interplay between plant architecture and climatic change. OpenAlea is a Python open source project (openalea.gforge.inria.fr) that provides support to a large community of users and developers.

2. USE CASE

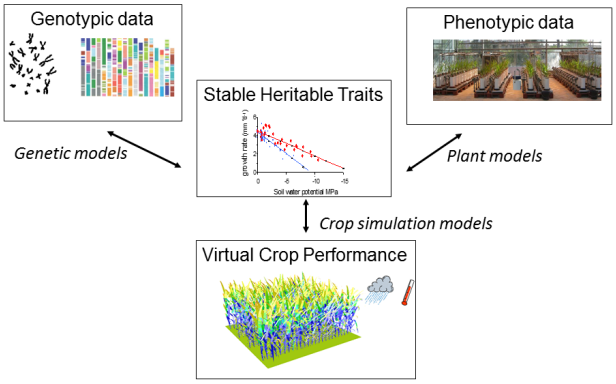

Figure 1: Use case

We consider a use case in the context of crop plant breeding [16], where high throughput data analysis needs to deal with simulation models at different scales (genes, organism and population). The objective of a breeding program is to produce plants that perform better than others (higher or more stable yields) in a given environment. This is challenging as environmental conditions vary a lot among cropping areas, and are subjected to rapid change in a global warming context.

Model-assisted breeding [10, 14] aims at tackling this issue. It combines plant models, which reduce phenotypic plasticity (that is, the response to external environmental changes), to a set of environment-independent plant parameters (stable traits), genetic models that link genetic profile (allele set) to stable traits, and simulation models that run virtual experiments to predict crop performance in a large set of environmental conditions (e.g., different light exposure, hydric conditions, nutriments...). Traits and plant parameters (such as the size of the plant, the length of its leaves, the number of tillers, ...) depend both on the plant genetic profile and on the response of the plant to environmental conditions, expressed as its phenotypic plasticity.

Model-assisted breeding recently gained interest thanks to the development of automated phenotyping platforms that allow the measurement of plant traits for a large number of accessions in controlled conditions. For example, the M3P-PhenoArch facility[1] allows to characterize daily plant growth and transpiration for 1,600 individuals at a time, together with precise control of water availability to the plants. It respectively generates 52 GB of data per day, 2.75 TB per essay (for a typical 50 days experiment) and 11 TB per year.

Managing such experiments is particularly challenging due to the volume of data involved, and the multi-disciplinary nature of the tasks, as it requires biological data production, data analysis, mathematical and biological modeling, and computer simulation. From a scientific workflow perspective, the main issue remains to combine (model-assisted) data analysis and model simulation with retro-action. In the use case, the simulated model is the growth of a set of plants, driven by environmental conditions (i.e. light and temperature). The growth of each plant depends on its parameters (plant traits), the environmental conditions modified by the other plants that compete for resource acquisition. The retro-action is due to the relationship between structure and function: the plant growth is impacted by the amount of light intercepted by the plant while the light intercepted by the plant, to its turn, depends on the plant growth. As a consequence, analyzing plant response to light first requires to estimate light amounts intercepted by each individual plant during their growth, using light simulation models and 3D plant reconstruction. Then, a light-driven growth model is to be inferred by fitting it to the observations. Finally, simulations allow to study how the light-growth feedback loop operates in a larger range of environmental conditions.

3. OPENALEA

This section introduces the OpenAlea system, with its programming and execution models.

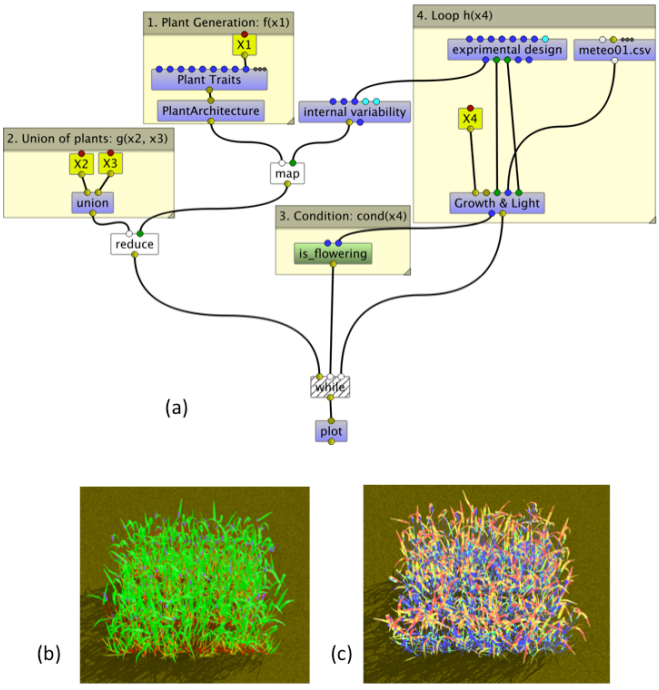

Figure 2: (a) OpenAlea workflow for simulating Maize and Wheat crop performance based on phenotypic and environment data, and two image outputs (b and c). Colors represent the organ's type in (b) and the amount of intercepted light in (c).

[1]http://www6.montpellier.inra.fr/lepse/M3P

Actors and workflows.

An actor in OpenAlea is an elementary brick (a.k.a. component or activity) that has a name, a function object (a functor, a program, a web service or a composite actor), and explicitly defined input and output ports. A semantic type [1] is associated to each port (with a corresponding color).

A workflow is represented as a directed multi-graph where nodes are actors, and directed edges are data links between output and input ports (see Figure 2(a)). A workflow can become a (composite) actor in another workflow to allow composition.

Dataflow variable.

One of the major originality of OpenAlea lies in the way iteration is handled by introducing a specific kind of actor, called *dataflow variable* X. It allows to specify that, at a given port, an actor receives an unbound variable rather than a value. Connecting an X to an actor transforms a workflow into a lambda function, and allows to express higher-order programming providing control flow behavior using a set of algebraic operators. The three iteration types can be expressed as [7, 5]: (1) counting loops without dependencies (*map* operator), (2) counting loops with dependencies (*reduce* and *for* operators) and (3) conditional loops (*while* operator). In Figure 2(a), the dataflow variables and the algebraic operators are represented using yellow and white nodes, respectively.

Execution (model-driven).

Dataflow execution in OpenAlea is orchestrated in a *model-driven* manner (rather than input-driven): the execution of a given workflow is launched in response to requests for data of one of its actors. Such an actor can satisfy the request when the upstream subworkflow has been executed, that is, when all the relevant actors connected to its input ports have been executed. When such an actor has received its data on its input ports, it executes and places data on its output ports. OpenAlea is able to deal with extremely large datasets to perform big data analysis in parallel environments.

Additionally, it allows actors to be *lazy* and *blocked*. When an actor is blocked, the execution is not propagated to the upstream subworkflow and when the actor is lazy, the execution is performed only if the actor's inputs have not changed compared to its previous execution. This type of orchestration performs only the operations needed to produce the required result, executing the subset of the graph relevant to the output [3].

Algebraic operators and λ-dataflow evaluation.

An algebraic operator is an actor that iterates over first-order function calls, and thus takes one or more functions as inputs. Ports that require a function have an associated semantic type *Function* (colored in white). For instance, the first input port of the *map* and *reduce* operators requires a function as input (see Figure 2.(a)).

λ-dataflow evaluation differs from the classical evaluation when the workflow contains at least one *dataflow variable* X. The execution is then decomposed into two stages. First, for each port of type *Function*, a subworkflow is computed if the upstream subworkflow contains at least one dataflow variable. This subworkflow is defined by all the actors needed to produce the data on this port, *i.e.* the upstream subworkflow and the connected output port. This subworkflow is dynamically transformed into a function (*i.e.* an actor) of one or several variables corresponding to its dataflow variables. Second, the evaluation of this function by algebraic operators consists in replacing the variables by real data and evaluating the subworkflow using the model-driven algorithm.

Reproducibility.

OpenAlea allows to make experiments reproducible by providing two capabilities. First, it is able to capture both prospective and retrospective provenance (following the PROV-DM model[2]), that is, it is equipped of a provenance module that keeps track of the complete description of the workflows as well as the full history of the data produced and consumed during each execution.

Second, and very originally, OpenAlea's architecture is based on IPython and makes use of IPython notebooks [19] to generate executable papers (see Figure 4). More precisely, OpenAlea workflows can be executed within IPython notebooks, through a web interface. Workflow results (including 2D plots, 3D scene graph, mathematical equations...) can be displayed within the notebook document and be shared with other users.

4. DEMONSTRATION

This section describes the main points of our demonstration.

We consider the workflows depicted in Figure 2 and 3 which implement the use case introduced in Figure 1. More precisely, the step *Stable Heritable Traits* of the use case is implemented by the module entitled *Plant Traits* in the workflow of Figure 2. A virtual crop is then designed (output of the *reduce* module). The crop growth is simulated and its performance assessed using a light interception model (implemented by the module *Growth & Light*). As for the step *Virtual Crop Performance* of the use case, it is evaluated by the amount of intercepted light at flowering time, still computed by the workflow of Figure 2. This workflow is reused as a composite module entitled *virtual experiment* in Figure 3, allowing to explore the *genotypic variability* by modifying the *Plant Traits*. Finally, both *Genotypic data* and *Phenotypic data* are taken into account to simulate *Virtual Crop Performance* for a large range of traits.

In our demonstration, we show how users can create or interact with highly expressive workflows (able to perform analysis, modeling and simulation tasks), both using the visual programming environment (Figure 3) and the IPython notebooks (Figure 4) of OpenAlea.

Reusing or designing a workflow.

OpenAlea offers a visual programming environment where users are provided with a set of predefined workflows and libraries of tools to be combined to form new workflows (see the left part of Figure 3, "Package Panel"). Users can create new wrapped tools by implementing them in Python. Each tool and workflow is associated with some documentation and saved. Ports of actors are typed and widgets can be associated with data types to allow users interacting with the data (see the widgets depicted in Figure 3.

[2]http://www.w3.org/TR/prov-dm/

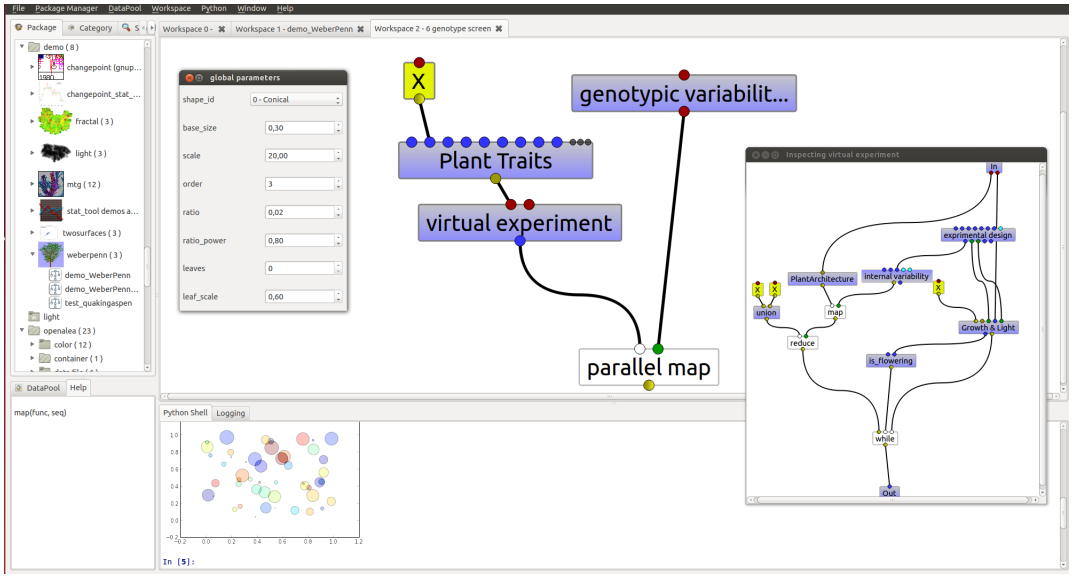

Figure 3: OpenAlea visual programming environment

(Re)Running a workflow.

To execute a workflow, users have to click on their output of interest (as OpenAlea is *model-driven*). For instance in Figure 2, by clicking on the *plot* actor, users trigger the execution of all the actors of the workflow, from top to bottom.

If users click again on any actor of the same workflow, they can visualize or access to intermediate results. OpenAlea determines whether or not any calculation has to be redone (default Lazy mode). If no input or parameter change has occurred, data is not recomputed. Otherwise, the subworkflow impacted by the change is executed again. In Figure 2(a), the *is_flowering* actor is a non-lazy or eager actor, colored in green. It is always recomputed, even if its input data has not changed.

Using algebraic operators for simulation.

Algebraic operators are higher-order actors that take function as argument. In our demonstration, we use three different operators: `map`, `reduce` and `while`. Other types of algebraic operators in OpenAlea follow the same principle while users can define their own operators.

The `map` operator is a higher-order function $map :: (\alpha \rightarrow \beta) \rightarrow [\alpha] \rightarrow [\beta]$. Its argument are a function $f :: \alpha \rightarrow \beta$ (first port) and a set of elements of type α (second input port). The `map` operator applies f to each element of the set and returns the set of resulting elements of type β.

Similarly, the `reduce` operator takes a function g of two variables and a sequence of elements $[x_i]$ and returns one element. `while` is an iteration operator that takes three inputs: an initial element t_0, a boolean function *cond* and function h. It initializes a variable t with t_0 and iteratively applies the function h on t while $cond(t)$ is true.

In the workflow in Figure 2, the `map` actor takes a λ-subworkflow f and a sequence of parameters $\mathcal{S}$. The λ-subworkflow f, composed of two actors (*Plant Traits* and *PlantArchitecture*), takes a parameter set that corresponds to one plant trait (*e.g.*, leaf growth dynamic) and generates an object that represents a fully parameterised individual plant model to be simulated. The sequence $\mathcal{S}$ is produced by the actor *internal variability*, and represents the intra-genotype (inter-individual) variability of the trait. During the execution, the `map` actor produces a sequence of individual plant models.

The `reduce` operator concatenates this sequence of plants into one graph corresponding to the crop canopy. Finally, the `while` operator simulates the development of the crop by iterating a growth function, that takes into account environmental data (meteo01.csv), the state and the specific parameters of each plant and the light intercepted by each 3D organs. The later is computed from the 3D geometry of the canopy. The simulation stops at the flowering stage.

Last, this workflow is reused as a composite workflow (see Figure 3) and run on a large set of genotypes to select the most efficient plant variety in a given environment.

From workflow to executable paper.

Execution of OpenAlea workflows can be embedded into IPython notebooks (Figure 4), able to produce *executable papers*, where users can share, visualize and interact with input and output produced by each step of an in-silico experiment in a web-based application.

5. CONCLUSION

Faced with the need of coupling data analysis with modeling and simulation, OpenAlea provides a unique solution able to extend the dataflow model of computation by introducing higher-order language constructs in a visual programming environment. Introducing first-class functions allows to design highly expressive workflows in a fully uniform way. First-class functions are increasingly popular and have also been introduced in several imperative languages like PHP, VisualBasic, C# or C++.

As for related work, considering Functional Programming in the context of scientific workflow systems [2, 20, 11] is not new and the number of solutions taking this direction has even increased in the last years. Functional Programming coupled with workflows is mainly used to reach to kinds of

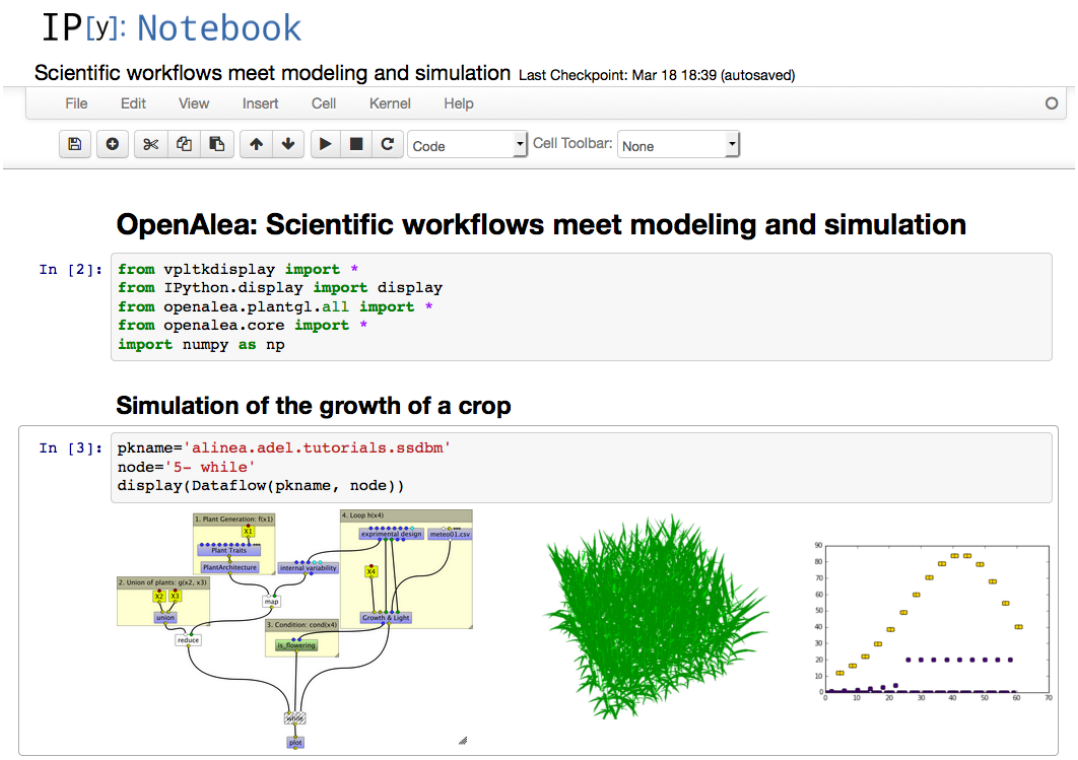

Figure 4: OpenAlea IPython notebook

goals.

First, Functional Programming is used to represent and formalize the semantics of workflows and their relationships with their executions. Interestingly, Kelly *et al.* [11] have introduced the λ-calculus (new) model of computation (MOC). In this context, authors have actually demonstrated that side-effect free workflow models can be defined as a subset of Functional Programming. Functional languages have been used to formalize workflow models in concrete workflow systems: this is the case in the Ptolemy II [13] system but also in the Taverna system where the semantics of Taverna's workflows have been recently rethought in functional terms [20].

Another (possibly complementary) aim to achieve when using Functional Programming is to deal with high-level data parallel structures. This is the case of the very recent *Cuneiform* system [2], which works on the Hi-WAY platform based on Hadoop YARN.

Like Cuneiform, the aim of the OpenAlea system is to exploit high-level data parallel structures. However, we want our system to be directly usable by end-users who are not computer scientists. Our originality here thus lies in allowing the use of functional programming and higher-order construct within a visual programming environment, as a mean to express control-flow constructs.

While OpenAlea is in used since 2007 (160,000 downloads, 1,200 distinct visitors a month, 20 active developers) leading to several biological findings (e.g., [12]), this paper is the first to provide an overview of the major capabilities of the OpenAlea system and the first to introduce the λ-dataflow concept.

This demonstration deals with the study of plant response to climatic change illustrating the research challenges in areas of high and increasing interest including *big data analysis* and *reproducible science*.

Acknowledgements

This work has been done in the context of the Computational Biology Institute (http://www.ibc-montpellier.fr). It has been partly funded by the ProvRecFlow project.

6. REFERENCES

[1] S. Bowers, B. Ludascher, A. H. Ngu, and T. Critchlow. Enabling scientific workflow reuse through structured composition of dataflow and control-flow. In *ICDE Workshops*, pages 70–70. IEEE, 2006.

[2] J. Brandt, M. Bux, and U. Leser. A functional language for large scale scientific data analysis. In *BeyongMR, ICDT/EDBT Workshop*, 2015.

[3] V. Curcin and M. Ghanem. Scientific workflow systems-can one size fit all? In *Proc. of Biomedical Engineering Conference*, pages 1–9, 2008.

[4] S. B. Davidson, S. C. Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire. Provenance in scientific workflow systems. *IEEE Data Eng. Bull.*, 30(4):44–50, 2007.

[5] J. Dias, G. Guerra, F. Rochinha, A. L. Coutinho, P. Valduriez, and M. Mattoso. Data-centric iteration in dynamic workflows. *Future Generation Computer Systems*, 2014.

[6] J. Dias, E. Ogasawara, D. De Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Algebraic dataflows for

big data analysis. In *Proc. of IEEE Big Data*, pages 150–155, 2013.

[7] E. Elmroth, F. Hernández, and J. Tordsson. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Computer Systems*, 26(2):245–256, 2010.

[8] J. Freire, C. Silva, S. Callahan, E. Santos, C. Scheidegger, and H. Vo. Managing rapidly-evolving scientific workflows. *Proc. of IPAW*, pages 10–18, 2006.

[9] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[10] G. Hammer, M. Cooper, F. Tardieu, S. Welch, B. Walsh, F. van Eeuwijk, S. Chapman, and D. Podlich. Models for navigating biological complexity in breeding improved crop plants. *Trends in plant science*, 11(12):587–593, 2006.

[11] P. M. Kelly, P. D. Coddington, and A. L. Wendelborn. Lambda calculus as a workflow model. *Concurrency and Computation: Practice and Experience*, 21(16):1999–2017, 2009.

[12] M. Lucas, K. Kenobi, D. Von Wangenheim, U. Voβ, K. Swarup, I. De Smet, D. Van Damme, T. Lawrence, B. Péret, E. Moscardi, et al. Lateral root morphogenesis is dependent on the mechanical properties of the overlaying tissues. *Proc. of the Nat. Academy of Sciences*, 110(13):5229–5234, 2013.

[13] B. Ludäscher and I. Altintas. On providing declarative design and programming constructs for scientific workflows based on process networks. 2003.

[14] C. D. Messina, D. Podlich, Z. Dong, M. Samples, and M. Cooper. Yield–trait performance landscapes: from theory to application in breeding maize for drought tolerance. *Journal of Experimental Botany*, 62(3):855–868, 2011.

[15] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In M. Gertz, T. Hey, and B. Ludaescher, editors, *Proc. of SSDBM*, Heidelberg, Germany, 2010.

[16] B. Parent and F. Tardieu. Can current crop models be used in the phenotyping era for predicting the genetic variability of yield of plants subjected to drought or high temperature? *J. of Experimental Botany*, 65(11):6179–6189, 2014.

[17] C. Pradal, S. Dufour-Kowalski, F. Boudon, C. Fournier, and C. Godin. Openalea: a visual programming and component-based software platform for plant modelling. *Functional plant biology*, 35(10):751–760, 2008.

[18] G. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig. Ten simple rules for reproducible computational research. *PLoS comp. biology*, 9(10):e1003285, 2013.

[19] H. Shen. Interactive notebooks: Sharing the code. *Nature*, 515(7525):151–152, 2014.

[20] D. Turi, P. Missier, C. Goble, D. De Roure, and T. Oinn. Taverna workflows: Syntax and semantics. In *e-Science and Grid Computing, IEEE International Conference on*, pages 441–448. IEEE, 2007.

Chapitre 5

Application : Un modèle géométrique de feuilles de graminées en croissance

Ce chapitre présente un modèle dynamique de feuilles de graminées pouvant être réutilisé dans différents modèles structure-fonction.

L'architecture des graminées, à la différence des arbres, est beaucoup plus définie par la forme des feuilles que par la ramification et l'arrangement des axes. La géometrie et le développement de la feuille de graminée peuvent être décomposées en différentes étapes, influencées par différents processus.

Ce chapitre illustre comment une source de connaissance (un modèle mathématique de feuille de graminée) peut être isolée d'un modèle structrure-fonction, afin d'être réutilisée dans différents contextes applicatifs.

Dans notre cas, nous sommes parti du modèle ADEL (Fournier et Andrieu, 1998). C'est un des premiers modèles structure-fonction permettant de simuler et de reconstruire la croissance et le développement du maïs. Il a ensuite été adapté au blé (Adel-Wheat Fournier et al., 2003), puis il a été étendu pour modéliser différents processus chez les céréales, comme la propagation des maladies (Robert et al., 2008 ; Garin et al., 2014 ; Garin et al., 2018 ; Robert et al., 2018).

A partir des connaissances acquises sur les feuilles de graminées et des limites observées dans la modélisation de la forme et de sa plasticité, nous décrivons un modèle 3D de feuille dynamique et plastique. Ce modèle permet de reconstruire des surfaces estimées à partir de données. La forme des surfaces sont alors déformées à l'aide de fonctions pour modéliser des stress environmentaux. De plus, un opérateur de simplification de maillage a été défini, non pas à partir de la surface maillée 3D, mais à partir de courbes 2D servant de génératrices de la surface. Cela nous permet de garantir, de façon efficace, la symétrie de la surface et de sa normale, condition importante pour la simulation des échanges radiatifs.

Ce modèle, bien que peu cité (7 citations source google scholar), est aujourd'hui utilisé dans le modèle Adele, intégré dans la plateforme OpenAlea. Il est aussi utilisé dans d'autres modèles d'OpenAlea comme Walter (Lecarpentier et al., 2019), pour modéliser des peuplements mixtes de blé ; OpenAlea.EcoMeristem pour modéliser le riz en 3D (Fournier et al., 2010), ainsi que pour d'autres applications comme la génération de données synthétiques pour le phénotypage haut-débit (Liu et al., 2017).

Ce chapitre est la version originale de l'article :

A Plastic, Dynamic and Reducible 3D Geometric Model for Simulating Gramineous Leaves

C. Pradal et C. Fournier
Publié à la conférence IEEE Fourth International Symposium on Plant Growth Modeling, Simulation, Visualization and Applications (PMA), 2012.

A Plastic, Dynamic and Reducible 3D Geometric Model for Simulating Gramineous Leaves

Christian Fournier
INRA
UMR 759 LEPSE
F-34090 Montpellier, France
christian.fournier@supagro.inra.fr

Christophe Pradal
CIRAD
CIRAD/INRIA/INRA EPI VirtualPlants, UMR AGAP
34398 MONTPELLIER CEDEX 5, France
christophe.pradal@inria.fr

***Abstract*—Unlike trees, the 3D architecture of gramineous plants is much more related to the shapes of its leaves than the arrangement of its branches. Many modelling efforts have thus concentrated on correctly capturing its complex shape at different stages and use them as scalable geometric primitives. Still, additional control of such objects is needed in the context of Functional Structural Modelling. The objective of this work is to propose a plastic and dynamic 3D leaf model that is well suited for such uses, still able to capture a variety of observed static shapes. Leaf shape is modeled by a parametric surface describing leaf midrib curvature, leaf width variation, undulation of leaf margins and twist along the midrib. Meshes can be generated from these surfaces, and reduced using a decimation algorithm. The model can be fitted with data or with curves drawn by user interaction. Morphological operators are defined and allows for plastic deformation of the control curves. The dynamics of shape acquisition can also be specified, and combined with morphological operators to simulate various scenarios of evolution and responses to stresses. The capabilities of the model are demonstrated through several cases of use. Future directions of research are thought to be a better integration of mechanical or physiological constraints that would reduce the model plasticity but avoid user-induced unrealistic simulation.**

***Keywords*-geometry; leaf; graminae; dynamic; plasticity; decimation; FSPM**

I. Introduction

3D geometry defines the interface of plants with the environment and is responsible for a great part of the visual realism of 3D reconstruction. For some plants, like trees, geometric models can concentrates on arrangement of branches and allometries between them without paying too much attention to the leaves, and obtain very realistic looking trees [1], [2]. Such an approach would on the contrary yield bad results on plants such as graminae where 3D architecture relies less on the arrangement of axes than on the individual shapes of leaves. This stands for representing plant shape at a given moment in time, but also for capturing its dynamic evolution during ontogeny or its changes in response to environmental stresses.

The geometry of the gramineous leaf and its dynamics can be decomposed in several parts that are determined by different processes (Figure 1).

Figure 1. Morphology of gramineous leaves exemplified by a growing maize plant (left) and top view of detached planified leaves (right). Leaf geometry can be decomposed in leaf width variation along the blade (right), leaf curvature at the midrib, and the undulation of margins. The top of the plant is composed by a whorl of rolled leaves, that modify their curvature.

First, meristematic activity at the base of the growing leaf results in leaf enlargement and extension, the balance between the two determining the variation of leaf width as a function of distance to leaf base [3]. This balance evolves during ontogeny and can be altered by the environmental conditions, especially light availability and N contents [4], [5]. On some species, like maize, meristematic activity in not homogeneous in the entire leaf section, and produces margins that have not the same length as the midrib region. This results in undulations at the leaf margins, or twisting of the principal plane of the leaf [6]. Second, the complex set of mechanical constraints that are exerted on leaf tissues during their maturation, together with the rate of tissue rigidification, will determined the local curvature of the leaf. This local component has been found to be the principal factor on maize leaves, the gravity only deforming to a small amount this rigid structure [7]. The dynamic of leaf curving during growth is additionally determined by the dynamics of leaf unrolling, which in turns depends of the unrolling rate of surrounding leaves in the whorl. Occasionally and for

short period, the leaf curvature can be strongly altered by stresses that induce rolling and straightening of leaves [8], [9]. Finally, gramineous leaves have at their base a tissue, the auricle, which can change its shape during ontogeny or in response to external conditions and actively modify leaf angle.

Embracing in a mechanistic model all this complexity is currently out of reach, and most of leaf geometric models are empirical and static. Still, some of the dynamic processes determining leaf shape, like leaf extension rate or the intensity of stresses, can be quantitatively predicted by functional models. Linking empirical static geometric models to dynamical variables in a realistic way may thus be challenging. In a 3D simulation, the level of geometrical details may also depend on the finality of the application. A detailed leaf (with undulation or twist) is needed for rendering realistic scenes, while simplified, still accurate, representation can help saving substantial computation time for e.g. light computation. A geometric model has thus to be modular enough to represent all the type of geometrical shapes, and perform efficient simplifications. The objective of this work is to propose an integrated solution for geometric modeling of gramineous leaves. We aim at building a plastic and dynamic 3D leaf model, capable of mimicking leaf dynamics but also able to simulate the elastic or non-elastic responses to stresses. We also conceived the model to capture a variety of shapes with different level of details and with capabilities of simplification in simulation. Finally, the model comes with methods for assimilation of usual static experimental data and derivation of dynamic parameters. To ease the diffusion of this model and allow its integration in several virtual gramineous plants, we design it as a software component available on the OpenAlea plateform [10].

II. Related work

The gramineous leaf geometry has been mathematically described for several aspects. The leaf width variation has been modeled, as a function of the distance to leaf base, by a polynomial [11], [12], [13], by modified polynomial function [14], or by a composition of functions reflecting leaf growth stages [15]. Leaf midribs have been described as arc of circle [16], arc of conics [13] or quadratics [17]. Leaf margin undulations have been proposed to be cycloid [18], [19], [20], [21]. These models are very compact and parsimonious, and give the user access to standard mathematical tools for modeling leaf shape transforms or analyzing derivatives. They have been used to reconstruct 3D plants from digitised partial data [22], [23], [20] or in models that concentrate on leaf extension [24], [25], [26], [27], [28]. In such cases, unknown parameters are considered as random variables that follow pre-defined or measured statistical laws [13], [20], [17]. Parameters could also be used for quantifying the genetic variability of leaf shapes [29]. This approach may however induced some inconsistency in the shapes produced, because they do not take into account the co-variation of parameters, and have a strong intrinsic constraint in the type of curve they can fit. For instance, the mathematical representation of the midrib can not represent a leaf midrib with inflection points.

A more flexible approach has been used to get 3D reconstructions of any type of leaves. The idea is to use smoothing function to build in a first step atlases of 3D shapes and use them, after appropriate scaling, to construct the plant.

This includes the use of smoothing spline [30], [23], [27] or hermit curves [31] that can be used to control advance graphics primitives like generalised cylinders [32]. In some cases, digitised mesh are directly used as scalable geometric objects [33], [34]. The drawback of the later method is that the modeler looses part of the control facilities offered by mathematical or control functions for operating on leaf shapes.

The methods described above stands for reconstructing plants at a given point in time (snapshots). The problematic is a bit different for dynamic simulations. First, dynamic simulation can be obtained by producing every time step a static reconstruction, possibly with evolving pre-defined shapes [16], [33], [35]. This approach has the advantage of simplicity, and gives satisfactory results at the canopy scale. Still, it does not allow predicting realistic patterns of shape evolution at the individual leaf or plant scale, as the system has no memory of the previous state. For example, re-scaling a pre-defined leaf object to account for a stress will effectively produce a smaller leaf, but with a completely new shape locally. Such metamorphoses during leaf ontogeny are embarrassing for the production of realistic animation of plant development. Also, in cases where leaf shape modification has an effect on plant growth (e.g. via reduced interception of light), the model will not correctly simulate the dynamics of differentiation of plants within the canopy. Finally, by definition, pre-defined leaf shapes are not predictive, and thus could not be used for its validation.

To solve these issues, more realistic, locally controlled, scenarios of individual leaf shape acquisition have been proposed. [24] and [36] used pre-defined mature shapes to build a dynamical scenario of evolution that truncate leaf width profile and use the mature leaf curvature as a trajectory for leaf midrib during extension. [37] proposed a set of rules for building growing leaves in the whorl as intermediate shapes between straight vertical ones and the shape of the first mature leaf. [38] defined a set of morphological operators that evolves with time and allow smooth simulation of individual leaf growth.

Whilst being compatible the every time step reconstruction strategy, our model clearly ambition to improve and ease the realistic modeling of individual leaf shape dynamic using time dependant operators. We also keep the idea of controlled curves, a method that both allow to fit to a large

range of data and keep mathematical control of the shape during growth.

III. Model description

A. Equation of the surface

Let $\mathcal{N}$ be a 2D curve representing the planar leaf midrib, and r be a function from $[0,1]$ representing the leaf width variation. The leaf shape is represented as a parametric surface constructed by sweeping a segment of length $r(s)$ along the midrib curve $\mathcal{N}$. The segment is orthogonal to the plane of the midrib curve.

Let $\mathcal{N} : s \mapsto (x(s), 0, y(s))$ and $r : s \mapsto r(s)$ be two functions from $[0,1]$ to $\mathbb{R}^2$ and $\mathbb{R}$ respectively. The absolut frame is defined as $(O, \vec{i}, \vec{j}, \vec{k})$. Let $\mathcal{R}$ be a rotation frame, also called moving frame [39], along a curve. The parametric surface obtained by sweeping a centered segment of length $r(s)$ along the midrib can be written as:

$$S(s,u) = \mathcal{N}(s) + \mathcal{R}(s)P(s,u) \tag{1}$$

If we consider $\mathcal{R}(s)$ the identity matrix and $\mathcal{P} : s, u \mapsto r(s)u\vec{j}$ a segment of length $r(s)$, then the equation is equivalent to :

$$\begin{aligned} &\forall s \in [0,1], \forall u \in [-0.5, 0.5] \\ &S(s,u) = (x(s), r(s)u, y(s)) \end{aligned} \tag{2}$$

To represent the twist of the leaf, the segment can be rotated along the tangent of the curve. The analytical form becomes more complex but quite similar to the previous one. In the same way, marginal undulations can be modelled by a different cycloid for each side of the leaf.

In the following equation, the segment is rotated around the y axis ($\vec{j}$) of an angle $\theta(s)$:

$$\mathcal{R} : s \mapsto (cos(\theta(s)), 0, sin(\theta(s))) \tag{3}$$

The equation becomes:

$$\begin{aligned} S(s,u) = (&x(s) + r(s)cos(\theta(s))u, \\ &r(s)u, \\ &y(s) + r(s)sin(\theta(s))u) \end{aligned} \tag{4}$$

Finally, the general form of the equation is:

$$S(s,u) = (x(s) + f(s,u), h(s,u), y(s) + g(s,u)) \tag{5}$$

with f, g, and h three functions that allow to represent several effects such as twist and undulation.

B. Fiting the model with data

Irrespective of the targeted application (static reconstruction or dynamical simulation) the model has to be fitted with data describing one or several steps of leaf midrib curvature evolution and profile of leaf width variation. In both cases, these data are normalised, in order to ease subsequent re-scaling for representing different leaves or link to physiological age.

The data can be obtained directly from leaf scans and digitization [40], [41], or from already modeled shapes, by sampling modeled curves at small intervals. Data are stored as collections in an indexed atlas, to allow for organizing variation with leaf rank and/or leaf ontogeny.

Building the parametric surface requires to be able to interpolate x, y and r along the midrib at any distance from leaf base. To do so, we first unify the parametrisation of the three variables as a function of s, and fit a smoothing spline for each. Starting from raw (x_i, y_i) data points, corresponding curvilinear abscissa s_i comes with :

$$s_i = \sum_{j=1}^{i} ds_i = \sum_{j=1}^{i} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{6}$$

As r is naturally expressed as a function of s, it is easy to get all three function fully parametrised at all s_i. The normalisation of the curve is done both for length and leaf surface. Normalising midrib $\mathcal{N}$ by leaf length L gives:

$$\forall s \in [0, L], \mathcal{N}^{'}(s) = (\frac{x(\frac{s}{L})}{L}, \frac{y(\frac{s}{L})}{L}) \tag{7}$$

That is after operating a simple variable change, we obtained:

$$\forall s^{'} \in [0,1], \mathcal{N}^{'}(s^{'}) = (\frac{x(s^{'})}{L}, \frac{y(s^{'})}{L}) \tag{8}$$

Moreover, on normalised leaves, the leaf surface is given by:

$$surface_{leaf} = \int_0^1 r(t)dt \tag{9}$$

To get this surface equal to unit, $r(s)$ is thus normalised to:

$$\forall s \in [0,1], r^{'}(s) = \frac{r(s)}{\int_0^1 r(t)dt} \tag{10}$$

C. Dynamic equation of the surface

A dynamic equation of the leaf surface can be obtained from one $r(s)$ function combined with several midrib curves $\mathcal{N}^i(s)$ describing the temporal evolution of the curvature. To do so, data are normalised and leaves reconstructed using physiological age ([42]). Physiological age represents the progress of leaf in its development. For simulation of plants where mature length is known, this age can be approximated by $\frac{l}{L}$, l being the current leaf length and L the mature leaf length. Normalised $r(s)$ gives $r(age)$. For leaf curvature, we considered a collection of normalised shapes $\mathcal{N}(age, s)$.

For reconstruction, $\mathcal{N}(s)$ at a given age is obtained by interpolating the different curves. The surface at t is given by progressing from the top of the leaf to the base.

Let L be the expected mature length, R be the expected maximal width, r_{max} the maximum of $r(s)$ and l the current leaf length. The surface is dependent of an unknown h:

$$\forall s \in [0,h], \forall u \in [-0.5, 0.5],$$
$$S^{'}(s,u) = (x(s)L, r(s+l-h)\frac{R}{r_{max}}u, y(s)L) \qquad (11)$$

The unknown h is solved numerically using:

$$\int_0^h \sqrt{\left(\frac{dx(t)}{dt}\right)^2 + \left(\frac{dy(t)}{dt}\right)^2}\, dt - \frac{l}{L} = 0 \qquad (12)$$

D. Differential form of the dynamic equation

The dynamic equation allows to fully parametrise a leaf emergence scenario for any stage of development of a leaf. However, it will produce always the same pre-defined mature leaf. The dynamic equation could also be written in a differential form, to allow dynamic transformation of leaf shape during a simulation. To do so, $r(age)$ is derived to get $\frac{dr}{dage} = f(age)$.

The variational midrib curve, $\mathcal{N}(age, s)$, is computing by interpolating a set of midrib curves with a mature length at different ages. It can be expressed either directly or by its derivative form:

$$\theta(age, s) = \frac{\partial \mathcal{N}}{\partial s}(age, s) \qquad (13)$$

At every time step of the simulation (dt), the user will define independently several functions:

- The rate of progression of the leaf through its potential development curve ($\frac{dage}{dt}$);
- The length increment ($\frac{dl}{dt}$);
- The variation of the radius ($\frac{dr}{dage}$);
- The variation of the curvature of the midrib ($\frac{d\theta}{dage}$).

E. Plastic transformations

The surface can be transiently transformed to mimic responses to environmental stress or the effects of constraints exerted by enrolling leaves. Operators can act on leaf curvature by modifying locally the amount curvature. First $\theta(s)$ is computed and the leaf is divided into intervals. A reduction of angle can be specified for every interval. Such operators can also apply on s and $r(s)$, and provide the user with another way of modeling stresses.

F. Surface computation and reduction

The surface equation can be evaluated at any number of points to build a detailed 3D mesh. Optional modifier of the surface (undulation and twist) can be de-activated to get a first reduction of the number of triangle.

We also implemented a reduction function that allows for further simplification whilst keeping surface unchanged and minimising simultaneously and optimally the error on x, y, r. This allows for keeping a maximum of principal characteristics of the surface (angles and width variation).

The algorithm takes as input the number of polygons desired for representing the surface. We do not use classic surface simplification algorithm (e.g. [43], [44]) but rather simplify the generative curve of the surface while minimising the error on the surface. Simplifying curves rather than surface is much more efficient. This is possible because we know the explicit equation of the surface and thus can compute the quadratic error on the surface when we remove a point on the curve. Moreover, simplifying the generative curves allow to maintain the symmetry of the surface while surface decimation algorithms had a tendency to break it, and thus modify the normal of the surface.

Algorithm is as follow :

- Over discretisation of the generative curve;
- Compute for all s_i the error induced if $x_i(s_i)$, $y_i(s_i)$ and $r_i(s_i)$ are discarded;
- Let $P_i = (x_i, y_i)$ and $R_i = (s_i, r_i)$ two set of points resulting for the discretisation of he curves. Let ε^{xy} and ε^{r} be the error on the $\mathcal{N}(s)$ curve and $r(s)$ curve respectively. The errors on each curves are given by:

$$\varepsilon_i^{xy} = distance(P_i, \overrightarrow{P_{i-1}P_{i+1}}) \qquad (14)$$
$$\varepsilon_i^{r} = distance(R_i, \overrightarrow{R_{i-1}R_{i+1}}) \qquad (15)$$

  The error on the surface is given by:

$$\forall i \in]1, n[, \varepsilon_i = \sqrt{(\varepsilon_i^{xy})^2 + (\varepsilon_i^{r})^2} \qquad (16)$$

- Select a point with the smallest error. Points are first added in a heap queue. When one point is removed, the error of its neighbors is updated and placed in the queue. The algorithm is reiterating until the number of points matches the requested number of points.
- Computation of the surface (triangulation) at the requested point and application of an optional deformation.

G. Implementation as components in OpenAlea

This package use PlantGL [45] for the 3D representation of the geometry of plants, and NumPy/SciPy packages [46] for interpolation and fitting. The model comes with simple visual interface and demos under VisuAlea (Figure 2).

Main functions are:

- Surface normalization and atlas construction from data
- Leaf element mesh builder with simplification given by one parameter
- Dynamic interface giving $shape = f(age, L, R)$
- Dynamic differential interface giving

$$dS = f(age, ds, dr, d\theta)$$

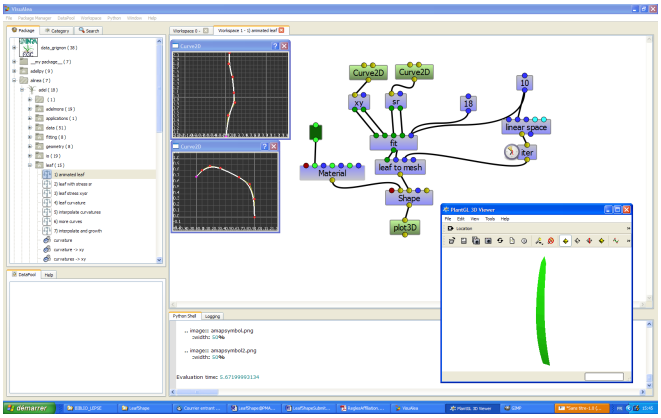

Figure 2. Snapshot of a visualea session running the model. Panels on the left allows for user-interaction in defing midrib curvature and leaf width profile. Leaf is then reconstruct, simplified and transformed into a mesh.

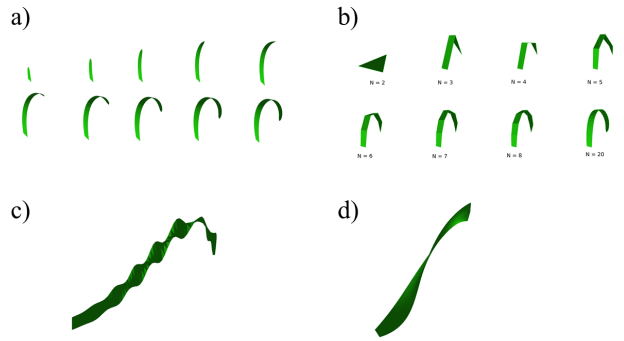

Figure 3. Illustration of some model capabilities. a) Dynamic evolution of the leaf surface; b) Simplification of a leaf with an increasing number of triangles but with the same leaf area; c) Undulation and d) Twist

- Plastic operators on s, θ, r
- Interface interval for applying stress factor along given s intervals

IV. Result

We will demonstrate the capabilities of our model by showing the plasticity of our leaves and use of the model in three case studies.

A. Plasticity of virtual leaves

Our model allows to build "multi-plastic" leaves as illustrated on Figure 3. First, our objects have a plasticity in the level of details at which a leaf is represented. Second the evolution of a leaf can be modeled. Finally, the use of morphological operators can add details or transiently affect the shape. The model was also able to capture a variety of shapes as illustrated by sketching of a variety of leaves (Figure 4) coming from a panel of numerised maize and wheat plants [47].

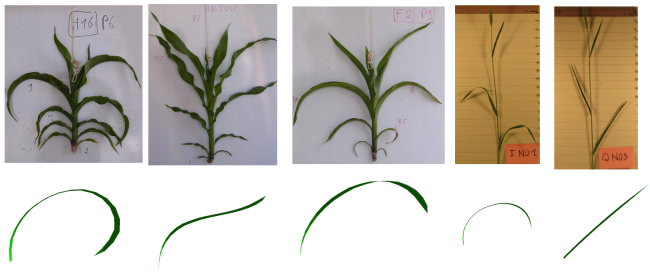

Figure 4. Sketching of different types of gramineaus leaves with the model: 3 varieties of maize contrasting in architecture and 2 varieties of wheat.

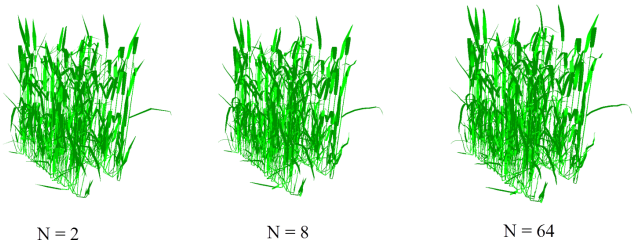

Figure 5. Reconstruction of a small wheat canopy using different levels of discretisation of leaves. Numbers indicate the number of polygon used per leaf.

B. Optimization of computing time

As stated above, the reducibility of the models allows to optimise computing time by reducing the number of triangle in a scene. We reconstruct with the ADEL-wheat model [27] a small patch of vegetation, using a measured set of leaves with curvature and leaf-width profile. Wheat is at maturity and leaves picked at random in an atlas. We then use the simplification utilities of our model to vary the number of polygons used to represent the leaves (every power of 2 between 2 and 128, Figure 5).

These mock-up were then used as input of a radiative transfert model, that allows computing light interception efficiency [48]. This model allows to replicate the patch, so that the computed value is for a whole canopy. Results show that computation time is a square function of the triangulation level, whilst light interception efficiency rapidly reach an asymptotic value (Figure 6). In this case, an optimal value would be 8 polygons per leaf.

C. Prediction of leaf shape deformation in response to stresses

Here we demonstrate how to use the model in a dynamic simulation where leaf shape can be dynamically modified during the simulation. We based on ADEL-maize [24], and test two hypothetic model that mimic the response to two stresses: a) leaf width is responding to local light illumination [5], and b) leaf extension rate and duration are responding to vapour pressure deficit in the air [49].

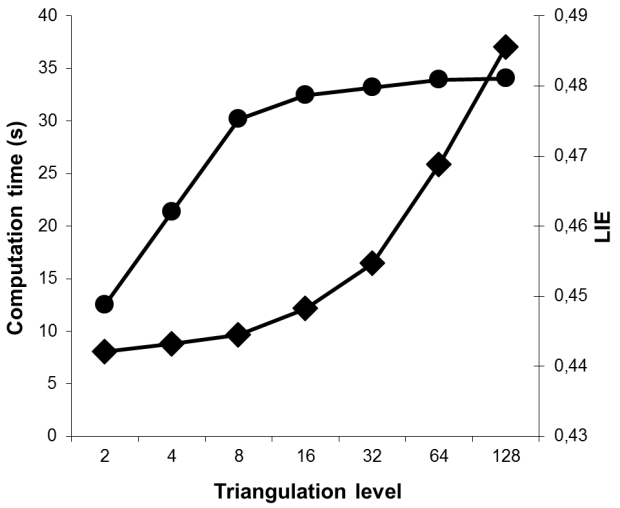

Figure 6. Computation time (diamonds) and light interception efficiency (circle) as a function of the triangulation level of the 3D scene.

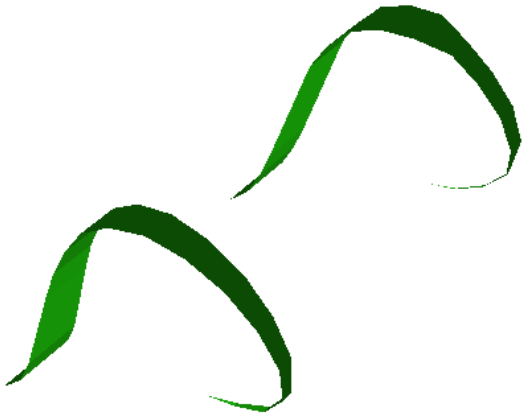

Figure 7. Prediction of leaf shape deformation as a function of response to stresses

Response to light can be modeled as a stress factor affecting the $r(age)$ curve. The response to vapour pressure deficit is modeled with a reduction of elongation rate and deceleration of the rate of progression of physiological age with time to mimic an extended duration of elongation. The model allows to predict a deformed shape that can be compared to experiment (Figure 7).

V. Discussion

We build an interactive, flexible and ready to use model of gramineous leaf shape. We show that it complies to previously used approaches for modelling leaf geometry in reconstruction or simulation applications. A first important difference is that it could do so all at once, i.e. without having to change the model itself to switch from an approach to another. A second important difference is that the model is not embedded into a particular plant model. We mainly demonstrate here the use with the ADEL family models [24], [27], but also we already use it with success on the rice model ecomeritem [50]. We also add in our package original functionalities, namely the simplification of shapes and a framework to include stresses occurring during growth. The main limitation of our approach is that the important flexibility implies a risk of simulating non realistic shapes. This is particularly true for the modeling of the evolution of curvature during leaf unrolling. Here we demonstrate the ability of the model to handle different scenarios, but all of these have to be parametrised. A further useful step would be to help the user with a less flexible parametrisation based on modeling of mechanical constraints. This step may be a key in our comprehension of gramineous plant development as leaf rolling dynamics within the whorl can mediate important retroaction determining whole plant architecture [51], [52], [31].

Acknowledgment

This project is supported by Agropolis Fondation in the frame of the OpenAlea project (project 0803-017) and by the Ministère de l'Ecologie et du Développement Durable in the frame of the Echap Project.

References

[1] J. Weber and J. Penn, *Creation and rendering of realistic trees*. ACM SIGGRAPH, 1995, p. 119128.

[2] B. Lintermann and O. Deussen, "Interactive modeling of plants," *IEEE Computer Graphics and Applications*, vol. 19, no. 1, pp. 56–65, 1999.

[3] B. Muller, M. Reymond, and F. Tardieu, "The elongation rate at the base of a maize leaf shows an invariant pattern during both the steady-state elongation and the establishment of the elongation zone," *Journal of Experimental Botany*, vol. 52, no. 359, pp. 1259–1268, 2001.

[4] C. Fournier, "Modélisation des interactions entre plantes au sein des peuplements. application la simulation des régulations de la morphogenèse aérienne du mas (zea mays l.) par la compétition pour la lumière," Thèse de doctorat, 2000.

[5] C. Fournier, B. Andrieu, and Y. Sohbi, "Virtual plant models for studying interactions between crops and environment," in *Simulation in industry, ESS'2001*, N. Giambiasi and C. Frydman, Eds., vol. Erlangen. SCS Europe Bvba, pp. 476–480.

[6] J. Ledent, T. Henkart, and B. Jacobs, "Phénologie du mas, visualisation de la croissance et du développement," *Revue de l'Agriculture*, vol. 43, no. 3, pp. 391–408, 1990.

[7] B. Moulia, M. Fournier, and D. Guitard, "Mechanics and form of the maize leaf: in vivo qualification of flexural behaviour," *Journal of Materials Science*, vol. 29, pp. 2359–2366, 1994.

[8] B. Moulia, "Leaves as shell structures: Double curvature, auto-stresses, and minimal mechanical energy constraints on leaf rolling in grasses," *Journal of Plant Growth Regulation*, vol. 19, no. 1, pp. 19–30, 2000.

[9] J. O. Hay, B. Moulia, B. Lane, M. Freeling, and W. K. Silk, "Biomechanical analysis of the rolled (rld) leaf phenotype of maize," *American Journal of Botany*, vol. 87, no. 5, pp. 625–633, 2000.

[10] C. Pradal, S. Dufour-Kowalski, F. Boudon, C. Fournier, and C. Godin, "Openalea: a visual programming and component-based software platform for plant modelling," *Functional Plant Biology*, vol. 35, no. 9/10, pp. 751–760, 2008.

[11] R. Bonhomme and C. Varlet-Grancher, "Estimation of the gramineous crop geometry by plant profiles including leaf width variations," *Photosynthetica*, vol. 12, no. 2, pp. 193–196, 1978.

[12] J. Sanderson, T. Daynard, and M. Tollenaar, "A mathematical model of the shape of corn leaves." *Canadian Journal of Plant Science*, vol. 61, pp. 1009–1011, 1981.

[13] L. Prévot, F. Aries, and P. Monestiez, "Modélisation de la structure géométrique du mas," *Agronomie*, vol. 11, pp. 491–503, 1991.

[14] J. Evers, J. Vos, C. Fournier, B. Andrieu, M. Chelle, and P. Struik, "An architectural model of spring wheat: evaluation of the effects of population density and shading on model parameterization and performance," *Ecological Modelling*, vol. 200, pp. 308–320, 2007.

[15] T. Dornbusch, J. Watt, R. Baccar, C. Fournier, and B. Andrieu, "A comparative analysis of leaf shape of wheat, barley and maize using an empirical shape model," *Annals of Botany*, vol. 107, pp. 865–873, 2011.

[16] T. Watanabe, J. S. Hanan, P. Room, T. Hasegawa, H. Nakagawa, and W. Takahashi, "Rice morphogenesis and plant architecture: Measurement, specification and the reconstruction of structural development by 3d architectural modelling," *Ann Bot*, vol. 95, no. 7, pp. 1131–1143, 2005.

[17] D. W. Stewart and L. M. Dwyer, "Mathematical characterization of maize canopies," *Agricultural and Forest Meteorology*, vol. 66, pp. 247–265, 1993.

[18] F. Aries, "Modélisation surfacique de la structure d'un couvert végétal pour l'étude du rayonnement," Thèse de doctorat, 1997.

[19] M. España, "Simulation de la variation temporelle, directionnelle et spectrale de la réflectance de cultures de mas partir d'un modèle dynamique de la structure 3d du couvert," Thèse de doctorat, 1997.

[20] M. España, F. Baret, F. Aries, and B. Andrieu, "Sensitivity of radiative transfer variables calculation to the accuracy of canopy structure description. the case of maize canopy as described by a 3d architecture model," *Agronomie*, vol. 19, pp. 241–254, 1999.

[21] M. L. España, F. Baret, F. Aries, M. Chelle, and L. Prevot, "Modeling maize canopy 3d architecture - application to reflectance simulation," *Ecological Modelling*, vol. 122, no. 1-2, pp. 25–43, 1999.

[22] N. Ivanov, P. Boissard, M. Chapron, and B. Andrieu, "Computer stereo plotting for 3-d reconstruction of a maize canopy," *Agricultural and Forest Meteorology*, vol. 75, pp. 85–102, 1995.

[23] J. L. Drouet, "Modica and modanca: modelling the three-dimensional shoot structure of graminaceous crops from two methods of plant description," *Field Crops Research*, vol. 83, no. 2, pp. 215–222, 2003.

[24] C. Fournier and B. Andrieu, "A 3d architectural and process-based model of maize development," *Annals of Botany*, vol. 81, no. 2, pp. 233–250, 1998.

[25] P. Kaitaniemi, P. Room, and J. Hanan, "Architecture and morphogenesis of grain sorghum, sorghum bicolor (l.) moench," *Field Crops Research*, vol. 61, pp. 51–60, 1999.

[26] P. Wernecke, G. Buck-Sorlin, and W. Diepenbrock, "Combining process- with architectural models: The simulation tool vica," *Systems Analysis Modelling Simulation*, vol. 39, pp. 235–277, 2000.

[27] C. Fournier, B. Andrieu, S. Ljutovac, and S. Saint-Jean, *ADEL-wheat: a 3D architectural model of wheat development*, ser. Proceedings - PMA03, 2003' International Symposium on plant Growth Modeling, Simulation, visualization and their Applications. Beijing, China, october 13-16, 2003. Beijing, China: Tsinghua University Press and Springer, 2003, pp. 54–66.

[28] J. B. Evers, J. Vos, C. Fournier, B. Andrieu, M. Chelle, and P. C. Struik, "Towards a generic architectural model of tillering in gramineae, as exemplified by spring wheat (triticum aestivum)," *New Phytologist*, vol. 166, no. 3, pp. 801–812, 2005.

[29] E. Ford, A. Cocke, L. Horton, M. Fellner, and E. Van Volkenburgh, "Estimation, variation and importance of leaf curvature in zea mays hybrids," *Agricultural and Forest Meteorology*, vol. 148, pp. 1598–1610, 2008.

[30] P. Lewis, *The botanical plant modelling system (BPMS)*. Thiverval-Grignon: INRA Unité EGC, 1997, pp. 45–53.

[31] A. Verdenal, D. Combes, and A. Escobar-Gutiérez, "A study of ryegrass architecture as a self-regulated system, using functional-structural plant modelling," *Functional Plant Biology*, vol. 35, no. 9/10, pp. 911–924, 2008.

[32] P. Prusinkiewicz, "Art and science for life: Designing and growing virtual plants with l-systems," *Acta Horticulturae*, vol. 630, pp. 15–28, 2004.

[33] Y. Guo, Y. Ma, Z. Zhan, B. Li, M. Dingkuhn, D. Luquet, and P. De Reffye, "Parameter optimization and field validation of the functional-structural model greenlab for maize," *Annals of Botany*, vol. 97, no. 2, pp. 217–230, 2006.

[34] C. Bassette and F. Bussiere, "3-d modelling of the banana architecture for simulation of rainfall interception parameters," *Agricultural and Forest Meteorology*, vol. 129, no. 1-2, pp. 95–100, 2005.

[35] A. Peyrat, O. Terraz, S. Merillou, and E. Galin, "Generating vast varieties of realistic leaves with parametric 2gmap l-systems," *The visual Computer*, vol. 24, no. 7, pp. 807–816, 2008.

[36] J. Ledent, *Generation of 3D representations of maize canopies from simple measurements : a tool for visualization or use with models involving plant architecture.* Beijing, China: IEEE computer society, 2006, pp. 282–285.

[37] M. España, F. Baret, M. Chelle, F. Aries, and B. Andrieu, "A dynamic model of 3d architecture: application to the parameterisation of the clumpiness of the canopy," *Agronomie*, vol. 18, pp. 609–626, 1998.

[38] N. Goel, L. Knox, and M. Norman, "From artificial life to real life : computer simulation of plant growth," *International Journal of General Systems*, vol. 18, pp. 291–319, 1990.

[39] W. Wang and B. Joe, "Robust computation of the rotation minimizing frame for sweep surface modeling," *Computer-Aided Design*, vol. 29, no. 5, pp. 379 – 391, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448596000772

[40] T. Dornbush, J. Watt, R. Baccar, C. Fournier, and B. Andrieu, *Towards a quantitative evaluation of cereal lamina shape using an empirical shape model*. Beijing, China: IEEE Computer Society, 2010, pp. 229–236.

[41] H. Sinoquet, B. Moulia, and R. Bonhomme, "Estimating the three-dimensional geometry of a maize crop as an input of radiation models: comparison between three-dimensional digitizing and plant profiles," *Agricultural and Forest Meteorology*, vol. 55, pp. 233–249, 1991.

[42] D. Barthélémy and Y. Caraglio, "Plant architecture: A dynamic, multilevel and comprehensive approach to plant form, structure and ontogeny," *Annals of Botany*, vol. 99, no. 3, pp. 375–407, 2007.

[43] M. Garland and P. S. Heckbert, *Surface simplification using quadric error metrics*. ACM SIGGRAPH, 1997, p. 209216.

[44] M. Garland and Y. Zhou, "Quadric-based simplification in any dimension," *ACM Trans. Graph.*, vol. 24, no. 2, pp. 209–239, 2005.

[45] C. Pradal, F. Boudon, C. Nouguier, J. Chopard, and C. Godin, "Plantgl: a python-based geometric library for 3d plant modelling at different scales," *Graphical Models*, vol. 71, no. 1, pp. 1–21, 2009.

[46] Scipy, *Scientific Tools for Python*. Enthought, 2012.

[47] P. Noblet, "Analyse de la variabilité génotypique de l'architecture 3d du mas," Mémoire de stage d'initiation la recherche, 2004.

[48] M. Chelle and B. Andrieu, "The nested radiosity model for the distribution of light within plant canopies," *Ecological Modelling*, vol. 111, pp. 75–91, 1998.

[49] K. Chenu, S. C. Chapman, G. L. Hammer, G. McLean, H. B. H. Salah, and F. Tardieu, "Short-term responses of leaf growth rate to water deficit scale up to whole-plant and crop levels: an integrated modelling approach in maize," *Plant, Cell & Environment*, vol. 31, no. 3, pp. 378–391, 2008.

[50] J.-C. Soulié, C. Pradal, C. Fournier, and D. Luquet, *Feedbacks between plant microclimate and morphogenesis in fluctuating environment: analysis for rice using Ecomeristem model coupled with 3D plant and energy balance computation tools in OpenAlea platform*. Plant Science Department, University of California, Davis, 2010, pp. 123–125.

[51] C. Fournier, J. L. Durand, S. Ljutovac, R. Schaufele, F. Gastal, and B. Andrieu, "A functional-structural model of elongation of the grass leaf and its relationships with the phyllochron," *New Phytologist*, vol. 166, no. 3, pp. 881–894, 2005.

[52] B. Andrieu, J. Hillier, and C. Birch, "Onset of sheath extension and duration of lamina extension are major determinants of the response of maize lamina length to plant density," *Annals of Botany*, vol. 98, no. 5, pp. 1005–1016, 2006.

Chapitre 6

Application : Modélisation structure-fonction pour la simulation d'épidémies foliaires

Ce chapitre présente un formalisme pour coupler un modèle structure-fonction avec des modèles de maladies folaires et de micro-climats.

Dans un contexte de développement durable et de changement climatique, il est important de trouver de nouvelles stratégies de protection des cultures, en diminuant les intrants. Pour cela, la modélisation permet de mieux comprendre les interactions majeures au sein du système dynamique qu'est la plante et peut amener à des stratégies de protection innovante. En particulier, les modèles structure-fonction ont été identifiés pour permettre d'optimiser l'utilisation de traits architecturaux (angle de feuille, dynamique d'apparition des feuilles et densité du feuillage) pour limiter la propagation des maladies. Cependant, du fait de la complexité de ce type de modèles, nous proposons un système conceptuel pour faciliter et étendre les modèles épidémiologiques aux modèles de plante structure-fonction.

Pour cela, un modèle formel est proposé décrivant les différentes entités et leurs interactions au sein d'un patho-système. Ce formalisme est implémenté au sein de la plateforme OpenAlea, en utilisant les concepts proposés dans cette thèse :

1. Un tableau noir, structure centrale partagée par les différentes sources de connaissances (architecture, processus micro-climatiques et maladies foliaires).
2. Un système de workflow scientifique permettant d'orchestrer la simulation de ce modèle complexe.
3. Une extension du modèle de calcul de dataflow avec évenement discret pour permettre d'ordonnancer des sources de connaissances s'exécutant à différents pas de temps.

4. Des sources de connaissances représentées sous la forme de composants logiciels réutilisables pour former des modèles modulaires.

Deux pathosystèmes contrastés ont été représentés et simulés : le blé / septoriose ainsi que la vigne / oïdium. La réutilisation est illustrée en réutilisant l'oïdium sur le blé, ainsi qu'en utilisant différentes stratégies possibles de propagation des maladies. Nous démontrons ainsi que nous pouvons réutiliser des connaissances développées dans différents domaines spécialisés dans des modèles complexes pluri-disciplinaires. Ce travail a été réalisé dans le cadre de la thèse de Guillaume Garin, dont j'ai été le co-encadrant principale sur la partie modélisation informatique.

Ma contribution dans ce travail a été de :

— Proposer un schéma conceptuel pour pouvoir modéliser un pathosystème en réutilisant des sources de connaissances existantes et hétérogènes, développées par des modélisateurs issus de différentes disciplines.
— Etendre le modèle de calcul aux évènements discrets pour prendre en compte différentes échelles de temps : la croissance de la plante, le développement du pathogène et la propagation des maladies s'effectuant à différents pas de temps.
— Proposer une abstraction à base de greffons permettant de modifier l'architecture et la maladie foliaire sans modifier le workflow scientifique, à partir d'interfaces communes.
— Simuler ce modèle complexe au sein de la plateforme OpenAlea.

L'article issu de ce travail a été cité 21 fois (source google scholar). Il est à la base de deux autres articles, dont un méthodologique, permettant la modélisation multi-échelles de complexes fongiques sur du blé (Garin et al., 2018). L'autre travail de recherche est une validation expérimentale du modèle sur blé/septoriose (Robert et al., 2018). Sur cette thématique, un projet européen vient de commencer. Il porte sur l'utilisation de ce modèle dans un cadre d'aide à la décision (projet H2020 IPM). Le code associé à ce travail est disponible sous license libre à l'adresse suivante `https://github.com/openalea/alep`.

Ce chapitre est la version originale du papier :

A generic functional-structural model to investigate the interactions between plant architecture, foliar pathogens and microclimate

G Garin, C. Fournier, B. Andrieu, V. Houlès, C. Robert et **C. Pradal**.
Publié dans le journal Annals of Botany, 114(4), pp 795-812.

Annals of Botany **114**: 795–812, 2014
doi:10.1093/aob/mcu101, available online at www.aob.oxfordjournals.org

ANNALS OF BOTANY
Founded 1887

PART OF A SPECIAL ISSUE ON FUNCTIONAL–STRUCTURAL PLANT MODELLING

A modelling framework to simulate foliar fungal epidemics using functional–structural plant models

Guillaume Garin[1,2,*], Christian Fournier[3], Bruno Andrieu[2], Vianney Houlès[1], Corinne Robert[2] and Christophe Pradal[4,5]

[1]*ITK, avenue de l'Europe, F-34830 Clapiers, France,* [2]*INRA, UMR 1091 EGC, F-78850 Thiverval-Grignon, France,* [3]*INRA, UMR 759 LEPSE, F-34060 Montpellier, France,* [4]*CIRAD, UMR AGAP and INRIA, Virtual Plants, F-34398 Montpellier, France and* [5]*Institut de Biologie Computationnelle, F-34095 Montpellier, France*
** For correspondence. E-mail guillaume.garin@itkweb.com*

Received: 29 November 2013 Returned for revision: 17 March 2014 Accepted: 28 April 2014 Published electronically: 12 June 2014

• **Background and Aims** Sustainable agriculture requires the identification of new, environmentally responsible strategies of crop protection. Modelling of pathosystems can allow a better understanding of the major interactions inside these dynamic systems and may lead to innovative protection strategies. In particular, functional–structural plant models (FSPMs) have been identified as a means to optimize the use of architecture-related traits. A current limitation lies in the inherent complexity of this type of modelling, and thus the purpose of this paper is to provide a framework to both extend and simplify the modelling of pathosystems using FSPMs.
• **Methods** Different entities and interactions occurring in pathosystems were formalized in a conceptual model. A framework based on these concepts was then implemented within the open-source OpenAlea modelling platform, using the platform's general strategy of modelling plant–environment interactions and extending it to handle plant interactions with pathogens. New developments include a generic data structure for representing lesions and dispersal units, and a series of generic protocols to communicate with objects representing the canopy and its micro-environment in the OpenAlea platform. Another development is the addition of a library of elementary models involved in pathosystem modelling. Several plant and physical models are already available in OpenAlea and can be combined in models of pathosystems using this framework approach.
• **Key Results** Two contrasting pathosystems are implemented using the framework and illustrate its generic utility. Simulations demonstrate the framework's ability to simulate multiscaled interactions within pathosystems, and also show that models are modular components within the framework and can be extended. This is illustrated by testing the impact of canopy architectural traits on fungal dispersal.
• **Conclusions** This study provides a framework for modelling a large number of pathosystems using FSPMs. This structure can accommodate both previously developed models for individual aspects of pathosystems and new ones. Complex models are deconstructed into separate 'knowledge sources' originating from different specialist areas of expertise and these can be shared and reassembled into multidisciplinary models. The framework thus provides a beneficial tool for a potential diverse and dynamic research community.

Key words: Functional–structural plant model, FSPM, OpenAlea, modelling foliar pathogens, multiscale tree graph, MTG, pathosystem, epidemic, septoria leaf blotch, *Septoria tritici*, *Mycosphaerella graminicola*, powdery mildew, *Uncinula necator*, wheat, *Triticum aestivum*, grapevine, *Vitis vinifera*.

INTRODUCTION

With incentives for more sustainable practices in crop protection it is important to decrease the usage of pesticides (Aubertot *et al.*, 2007). This implies that, rather than eliminating pathogens, crop protection lowers damage of pathogen origin to an acceptable level, by combining reduced chemical control with resistant cultivars and environmentally responsible agronomic practices. In turn, a better understanding of pathosystems is required. We use pathosystem to mean a dynamic ensemble consisting of a host plant population, a parasite population and their biophysical environment. Pathosystems involve multiple levels of interactions that are the source of complex behaviours.

In this paper we focus on interactions between crop structure, fungal foliar pathogens and microclimate. The canopy is the substrate and support of pathogen reproduction and dispersal (Ando *et al.*, 2005; Walters and Bingham, 2007). The dynamic nature of the canopy structure and microclimate makes it difficult to analyse host–pathogen interactions in field experiments (Lovell *et al.*, 1997). A modelling tool coupling crop and pathogen development might help to disentangle and quantify the interactions between the canopy structure, its pathogens and the environment (Prusinkiewicz, 2004; Lucas *et al.*, 2011). It could contribute to promote agricultural strategies of disease control through canopy properties (Baccar *et al.*, 2011; Gigot *et al.*, 2013).

Different models simulating epidemics and accounting for characteristics of the canopy have been developed. Some of these express the influence of major canopy features on disease dynamics (Burie *et al.*, 2011; Caubel *et al.*, 2012). These models use a limited set of parameters. The environment is averaged at coarse scale and interactions between plant and

pathogens are not localized in the canopy (Madden *et al*., 2007). In the model of Casadebaig *et al*. (2012), disease dispersal is influenced by plant architecture. The latter is simulated through integrative variables, which approximate the geometry of the canopy. Graph networks define the adjacency of plants. Yet the local organ environment is not taken into account for pathogen development.

Room *et al*. (1996) and Wilson and Chakraborty (1998) have proposed to use functional structural plant models (FSPMs) to simulate interactions between plant structure and epidemics. FSPMs simulate dynamically three-dimensional (3-D) plant architectures. Organ emergence, growth and death are described, associated with their precise localization. These features can be used to characterize the interactions between pathogens and the tissues they colonize, as well as spore dispersal from a localized inoculum source to other localized healthy plant tissues. The co-localization of variables of interest allows characterization of the plant–pathogen–climate interactions from the local scale (cm^2), where processes such as infection and tissue colonization are described, to the canopy scale, typically a few square metres for local dispersal.

With this in mind, some coupled FSPM–epidemic models have been developed with the main objective of better understanding dynamic interactions between the pathogens, their host and the environment (Calonnec *et al*., 2008; Robert *et al*., 2008; Pangga *et al*., 2011). These models include the effects of leaf age and size on the fungal infection cycle (Calonnec *et al*., 2008; Robert *et al*., 2008). They estimate the microclimate environment (Saudreau *et al*., 2007). Spore dispersal is simulated accounting for the distance and obstacles between healthy and sporulating leaves (Calonnec *et al*., 2008; Robert *et al*., 2008). For grape powdery mildew, simulations have revealed that differences in the positions of leaves in the canopy and in leaf susceptibility strongly influence the epidemic (Calonnec *et al*., 2008). The traits of wheat architecture influencing *Mycosphaerella graminicola* (anamorph: *Septoria tritici*) epidemics have been ranked (Robert *et al*., 2008). These models were also used to evaluate wheat ideotypes (Fournier *et al*., 2013) and to better understand the role of canopy architecture on the effect of sowing density on epidemics (Baccar *et al*., 2011). These modelling case studies have highlighted the potential role of FSPMs to study the effects of plant architecture dynamics on foliar pathogen epidemics.

However, only a small number of FSPM–fungus coupled models have been developed. One explanation for this lies in the high cost of building the conceptual framework and in their implementation as several sub-models (3-D plant model, an infection cycle model, a dispersal model and several physical models of microclimate). Each modelling solution available has proposed a particular implementation specific to one pathosystem. As suggested by Mammeri *et al*. (2010), a more generic solution could arise by considering the pathosystem model as a collection of interoperable software components (Pradal *et al*., 2008). We propose that the key processes of a wide range of foliar fungal pathosystems can be captured by a generic modelling framework, which would allow integration of the specificities of a given plant–pathogen couple with relatively little effort.

This framework will include models of plant growth and structure, but also development of the parasite population, which has a different biological form and life cycle, yet interacts with the plant. The challenges are to model two structurally different systems maturing in parallel and interacting on multiple spatial and temporal scales, and to define generalized communication rules between plant and parasite. The framework should (1) propose interfaces of communication between separated FSPMs, bio-physical models and fungal models, and (2) depict how to construct foliar fungal models to integrate them in pathosystem models with FSPMs and bio-physical models.

Multiscale modelling of these systems is at the crossroads of several scientific disciplines: plant pathology, ecophysiology, mathematics, physics and computer science. This requires combining different models and knowledge produced in these disciplines. Cieslak *et al*. (2011) have addressed this issue for plants, coupling structural models with functional models. Their solution, however, applies only to the restricted framework of L-systems (Cieslak *et al*., 2011). The platform OpenAlea provides an alternative solution, allowing the integration of models written in different computational languages (C, C++, Fortran, R, Python) in the form of distributable and interoperable software components (Pradal *et al*., 2008). All plant models (Fournier *et al*., 2003; Louarn *et al*., 2008) and physical models (Sinoquet *et al*., 2001; Robert *et al*., 2008) used in this study were already developed and integrated in the platform.

This paper is organized as follows. First, we present the pathosystem and the main interactions addressed in the framework. Second, we present the model with the conceptual choices and the key features of implementation. Third, we expose how two different models of pathosystems were implemented in the framework and we demonstrate the ability of the framework to test the impact of canopy architectural traits on fungal dispersal.

THE FOLIAR FUNGAL PATHOSYSTEM

The pathosystem comprises three entities: the canopy, the fungal population and the environment. General features of the two biological entities are described below. They have been used to build our modelling abstractions. The fungus and the canopy have direct interactions and they also interact with the environment, which therefore constitutes a medium of indirect interactions. These interactions form the core of our modelling framework.

The canopy

The canopy is composed of individual plants. Each plant can further be viewed as a colony of interconnected organs (White, 1979; Fournier and Andrieu, 1998; Vos *et al*., 2010). The geometry and the position of each organ in 3-D space determines its local interface with the environment (Chelle, 2005), and with the spores of the pathogens. The physiological tissue properties determine the constraints for fungal growth after infection.

Canopies are dynamic objects with dynamic structures (Giavitto *et al*., 2004). Plant functioning results in continuous changes of the internal state of plant organs, in terms of composition (e.g. water, nutrient), fluxes (e.g. transpiration, carbon, hormones) and enzymatic activity (e.g. photosynthesis).

Structural and geometric changes occur in a more discrete manner over time, with periods of growth or senescence alternating with periods of structural stability (Barthélémy and Caraglio, 2007). Structural changes consist of the production of new phytomers and branches, their extension and their eventual death

(Carbonneau *et al*., 2003). The dynamics of phytomer production and extension depends on temperature of meristematic regions (Parent *et al*., 2010), other environmental signals (photoperiod, light quality) and the internal state of the plant (e.g. amino acids and sugar). Organ death is also regulated by the environment and plant internal state. Because of the sensitivity of plant morphogenesis to environmental factors, plant morphology exhibits a high level of plasticity depending on growth conditions (Mech and Prusinkiewicz, 1996; Moulia *et al*., 1999; Barthélémy and Caraglio, 2007; Baccar *et al*., 2011).

Finally, plant and canopy geometry can undergo reversible changes at smaller time scales, in response to alternating environmental conditions (e.g. tropism, leaf rolling).

The fungal population

The fungal population is composed of individual spores and lesions, taking different forms, with colonizing or reproductive behaviours. The infection cycle is common to most foliar fungal species and is well documented in the literature (Rapilly, 1991; van Maanen and Xu, 2003; Caubel *et al*., 2012). Two biological forms of the fungus are identified: (1) lesions, which display as symptoms on leaves and sporulate to produce new dispersal units; and (2) the dispersal units, which are dispersed at the canopy level and infect the leaves. One dispersal unit represents either one spore (e.g. *Puccinia triticina* spores individually dispersed by wind; Eversmeyer and Kramer, 2000) or one aggregate of spores (e.g. *Septoria tritici* spores grouped in infectious droplets of rain; Gigot *et al*., 2013). One dispersal unit produces one single lesion after infection if specific environmental conditions are met.

During an infection cycle spores germinate and the fungus penetrates the leaf, thus forming a lesion that will sporulate and emit a new generation of dispersal units (Rapilly, 1991). After penetration, the fungus first undergoes a phase of latency characterized by continuous growth and tissue colonization, during which symptoms may appear but no spore is produced. After this latency period, the sporulation period, is the reproductive phase of the cycle, which is generally paced by discrete dispersal events occurring with rain or wind (van Maanen and Xu, 2003). An infection cycle usually lasts a few weeks, and numerous fungal generations can occur during a crop growing season (Bolton *et al*., 2008).

The growth of fungi is driven by local temperature. *Septoria tritici* achieves fastest infectious cycles around 18 °C, and does not grow at temperatures too low or too high (Bernard *et al*., 2013). Likewise, many pathogens can only infect the host leaves in the presence of free water [e.g. *Plasmopara viticola* on grapevine (Magarey *et al*., 2006); *Stemphylium botryosum* on lentil (Mwakutuya and Banniza, 2010)]. Other climatic variables have been shown to modify the fungal response, such as relative humidity [e.g. *Venturia inaequalis* on apple trees (Gadoury *et al*., 1998)] or radiation [e.g. *Uncinula necator* on grapevine (Austin and Wilcox, 2012)].

Effects of the canopy on foliar pathogens

In pathosystems, plants are the substrate for the fungus, and the canopy is the medium where its dispersal occurs. The host–parasite relationship is compatible when susceptible plant tissues are in the fungus' range of targets during periods of infection.

Plant characteristics have been shown to influence epidemic development in different ways:

(1) The size and physiological status of the plant tissue determine the resource availability for colonization and multiplication of the pathogens. The availability of tissues is a limiting factor to fungal colonization (Robert *et al*., 2004). Fungal compatibility with leaf substrate depends on their trophic behaviour of the fungus. For example, the biotrophic powdery mildew (*Uncinula necator*) is stopped by senescence due to a depletion of available nutrients (Galet, 1977). By contrast, *Mycosphaerella graminicola* is hemibiotrophic: penetration into the leaf and primary mycelium development occurs in living leaf tissues, followed by the appearance and growth of chlorotic symptoms, which develop into necrotic sporulating lesions (Robert *et al*., 2008). A further example is the strong influence of leaf physiological state (nitrogen status in Robert *et al*., 2004) on the production of spores of *Puccinia triticina*.

(2) Resistance mechanisms also stand out at the tissue scale. They may involve those relying on chemical and enzymatic reactions or are based on physical properties of the leaf surface. For example, grapevine leaves become less susceptible to infection by powdery mildew with ageing as the cuticle strengthens (Calonnec *et al*., 2008).

(3) Canopy architecture influences the microclimate in which pathogens develop. The climate within a canopy is heterogeneous (Chelle, 2005). Light, rain and wind penetration are a function of leaf area distribution and canopy height (Jones, 1992; Varlet-Grancher *et al*., 1993). Humidity of the air within the canopy depends on light and wind penetration and on plant transpiration (Tuzet *et al*., 2003), and leaf wetness depends on all these factors. Microclimatic factors on or around the leaves modify the response of parasites (Lovell *et al*., 2004; Bernard *et al*., 2013). In the sclerotinia–carrot pathosystem, lateral trimming after canopy closure has been shown to favour dryer micro-conditions, therefore reducing significantly disease pressure until harvest (McDonald *et al*., 2013).

(4) The spatial density of plant organs influences spore dispersal from infected organs to healthy organs (Calonnec *et al*., 2012). This was highlighted in septoria leaf blotch, upward dispersal of which is driven by rain splash, and is thus sensitive to rain penetration. Eyal (1971) noted that the introduction of dwarf varieties of wheat correlated with a sharp increase in the incidence of septoria leaf blotch. Then, Bahat (1980) and Lovell *et al*. (1997, 2004) showed that spread of the parasite to the top is faster if successive leaves are closer. Lovell *et al*. (1997) suggested a hypothesis linked to the stem extension speed.

(5) Dates of emergence and death of the plant organs determine the temporal synchronism between pathogen development and the organs that they colonize. A pathosystem is a dynamic system: its state at a given time results from the historical evolution of its interacting elements. However, plant functioning, canopy growth, microclimate dynamics and pathogens do not evolve at the same rhythms. This is an

important determinant of compatibility. Compatibility is only ensured if synchronism occurs between the period of sensitivity of the plant and the period of pathogenicity of the fungus. For instance, the period of flowering transition is favourable to many pathogens (Costes *et al*., 2013), such as *Sclerotinia sclerotiorum* on oilseed rape. This fungus primarily infects the flowers. In a second step, infectious petals become a vector of disease transmission when falling on the leaves (Young *et al*., 2007).

FRAMEWORK OVERVIEW

Formalization of the framework: concepts

Scope of the modelling framework. Our framework targets models of a pathosystem that comprises the functional–structural crop, the fungal population and the physical environment. The multi-scaled canopy is simulated with an FSPM. The latter can describe both the geometric development of the canopy and plant functioning at the level of individual organs (Parent *et al*., 2010; Vos *et al*., 2010). FSPM can be tightly connected to physical models computing the micro-environment below the organ scale (Saudreau *et al*., 2007, 2013). Various models of microclimate at the leaf scale have been developed using FSPM (Chelle, 2005): models for radiation and temperature (Dauzat *et al*., 2001; Sinoquet *et al*., 2001; Chelle and Gutschick, 2010; Ngao *et al*., 2013), models for rain interception (Bassette and Bussière, 2005), models of wind distribution in canopies (Tuzet and Wilson, 2002) and models of leaf wetness (Leca and Saudreau, 2010).

Explicit description of plant architecture in FSPM provides a fungus model with variables such as the size and age of individual organs (affecting the pathogen cycle) and the local light, temperature and humidity (impacting fungus development). With FSPMs, the plant–pathogen–climate interactions are expressed from the local scale (cm^2), where processes are described, to the square-metre canopy scale. Relationships occurring above (e.g. landscape) or below (e.g. cells) these scales are not considered.

In this study we did not implement new models of microclimate, nor did we specify how plant models should respond to climate or microclimate models. Our focus is on the model for the fungal population and its integration in the pathosystem model. The following section details (1) concepts for coupling a model of foliar fungus with other models in a functional–structural pathosystem: temporal orchestration and interfaces of interaction between models; and (2) abstract processes that comprise a model of foliar fungus in our framework.

Management of temporal scales. A first key issue for coupling two complex biological systems is to provide flexibility for managing different temporal scales. Modellers usually choose a larger time step for the plant (several degree-days or a few days) than for the fungus (1 h to 1 d) because of different length of life cycles. Moreover, plants and pathogens may not age under the influence of the same factors. For example, the continuous development of the plant may be paced by a sum of favourable temperatures, the growth of a fungus by a hydro-thermic time. In contrast, dispersal is computed only on discrete rainy or windy events. To manage this, the pathosystem is modelled as a discrete event system. The modeller defines the temporal scale for each process prior to the simulation and delegates to a scheduler the tasks of orchestrating the synchronization between processes.

Spatial scales of interaction in the framework. For the interface of fine interactions between the fungus and the plant, we introduce the notion of a '*phyto-element*'. We define a phyto-element as the smallest unit of leaf tissue explicitly modelled in the plant data structure at which local information, such as microclimatic variables, is aggregated. This definition is technically compatible with all arbitrary partitioning of plants into elements. However, for foliar pathogens, we would not recommend choosing phyto-elements larger than leaves, as this would result in loss of precision in model simulations. The location, geometry and topological relationship of the phyto-element to the rest of the canopy are managed by the plant model. The plant model also provides access to variables of interest above the scale of phyto-elements if needed (e.g. the total space available on a leaf divided into several phyto-elements, progress of senescence at the scale of the organ).

At the phyto-element, the fungus model can have access to quantitative and qualitative information about the plant tissues, and to micro-climatic variables such as temperature, rain or moisture (Fig. 1). In turn, the fungus colonizes space, consumes resources (N, C), possibly produces effectors and returns this information to the plant model. Although multiple forms of feedback from the disease to the plant were integrated from a conceptual perspective in the modelling framework (e.g. alteration of photosynthesis, competition for nutrients, response to effectors), the actual implementation was still limited to the following: colonizing lesions reduce plant photosynthetic area, which in turn influences the success of further infection and lesion growth.

The fungal population is simulated as a system of separate individuals (i.e. lesions and dispersal units) to possibly model intrinsic variability within a population. However, a phyto-element can carry several individuals that receive the same information and are together responsible for the emergence of symptoms at this scale. Each individual operates as an automaton passing through the epidemic cycle in a stepwise sequence according to local conditions. The epidemic cycle described in the previous section can be refined to be more specific for a particular disease.

A different kind of interaction occurs during the physical transport of dispersal units, where models operate on the geometry of the canopy at a large scale, and during which dispersal units are considered as atomic entities. In the range of spatial scales managed by our framework, we address short- and medium-range dispersal.

Description of the fungus model (Fig. 2)

Lesion. Each lesion is viewed as an automaton undergoing several developmental stages until sporulation (Fig. 2-1). For the transition between stages, we introduce the concept of physiological age of the lesion. The response to external factors (e.g. climate, availability of nutrients) is modelled with a particular rate of physiological ageing.

To handle more complex interactions with the plant such as the competition for shared resources, the lesion automaton can optionally process its update in two steps. In a first step, a cost of growth (which can be of different nature) is estimated for further processing by an external model, such as a nutrient-

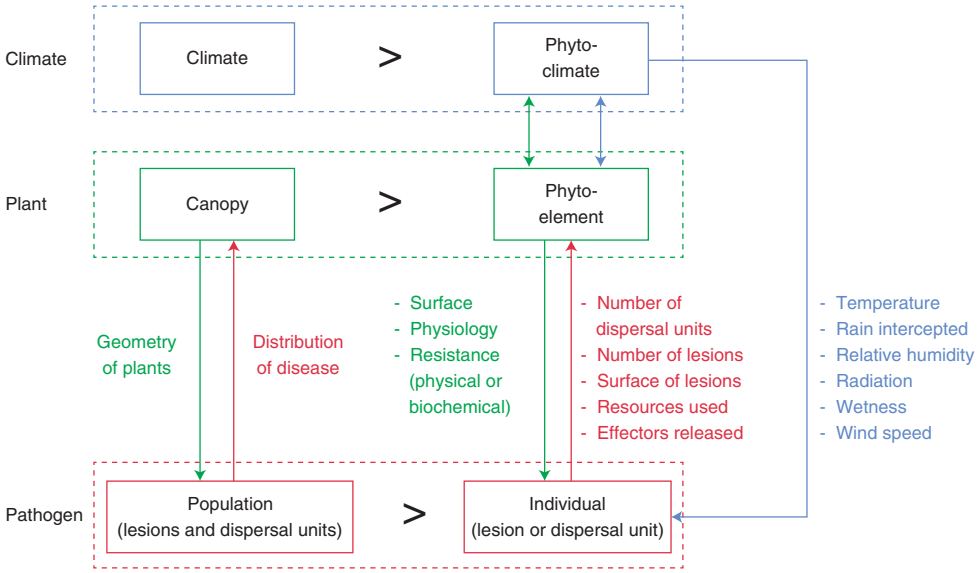

FIG. 1. Conceptual diagram of interacting entities in a pathosystem for fungal foliar diseases on plants. Blue: climate-related; green: plant-related; red: pathogen-related. The boxes indicate spatial scales, the larger scale being on the left (as indicated by the > symbol). Arrows: flows of information.

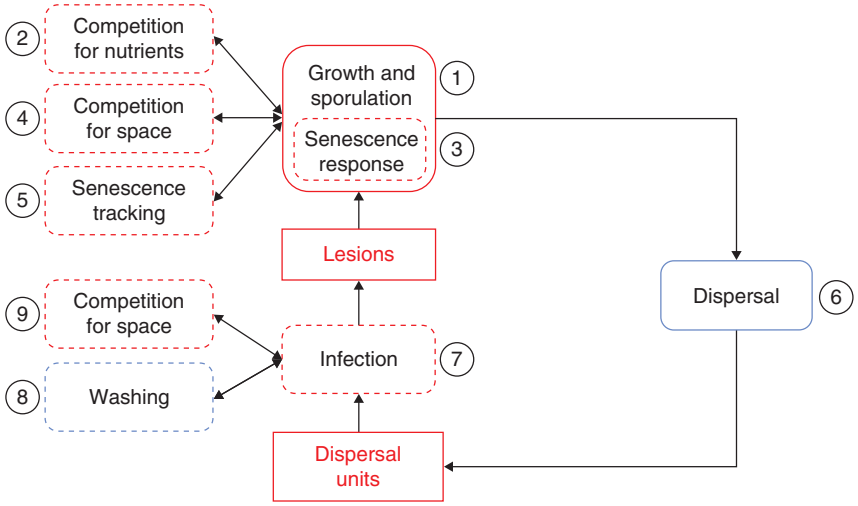

FIG. 2. Generic processes affecting fungal forms and the scale at which they operate. Boxes in red: individuals of the fungal population. Boxes with lower-case letters indicate processes that are simulated, with solid boundaries showing mandatory process and dashed boundaries showing optional process. For numbers refer to the text.

sharing model. Actual growth of the lesion is evaluated in a second step according to the response of the resource-sharing model (Fig. 2-2).

Moreover, lesions may optionally handle models for managing lethal response and evaluating internal damage (e.g. response to senescence) (Fig. 2-3).

A lesion internally manages its positioning on the leaf and an individual set of parameters that determine its behaviour (e.g. growth rate, thresholds for changes of developmental stages, shape parameters for response functions to microclimate). It communicates to the plant or to external models its damage on the leaf (surfaces and type of symptoms), its physiological age, its developmental stage and the stock of spores it produces.

Interactions between lesions. Lesions can interact with each other during development (competition for space, collective contribution to symptoms). These processes are not managed by individual lesions, but by (optional) external models.

Such models operate at the spatial scale of the phyto-element, in a similar way as models handling shared resources between the plant and the pathogen (Fig. 2-4). As input, they read all the growth demands of the lesions in place. As output, they redistribute a growth offer to each lesion relative to its growth demand, the area available on the phyto-element and, possibly, rules of priority between lesions of different ages.

When senescence is accounted for, external models indicate which lesions are senescent (Fig. 2-5). In such a case, the response method of the lesion is engaged.

Dispersal. Dispersal events occur in two phases: the emission separates the inoculum from the source phyto-element and the transport then distributes the inoculum on target phyto-elements. For emission, physical models separate spores from the lesions under the action of rain or wind, and distribute them in dispersal units for transport (Fig. 2-6). Models for rain-driven emission will aggregate several spores in infectious rain droplets. Models for wind-driven emission will generally consider a spore as a single dispersal unit.

Dispersal units emitted by the same phyto-element are considered as a unique source, without tracking provenance from individual lesions. Transport is also calculated by physical models and depends on canopy architecture.

Dispersal unit. At the leaf scale, the main function of a dispersal unit is to achieve infection (Fig. 2-7). After infection, one dispersal unit creates one lesion and disappears.

A dispersal unit is an object with the following unique attributes: its position on the leaf, the number of spores it carries, a set of parameters that are the same for all dispersal units of the same species (growth rate, thresholds for changes of developmental stages, shape parameters for response functions to microclimate, etc.), and parameters of its internal progression towards infection.

External models operating on dispersal units. Dispersal units that have not achieved infection can be manipulated by external models. For example, during rain, dispersal units can be washed off the leaves (Fig. 2-8).

Other external models that have information on the status of the phyto-element determine if particular dispersal units can infect the tissues beneath them (Fig. 2-9). This type of model can be made in a probabilistic way relative to free space available or according to the position of dispersal units if senescence is localized.

Architecture of the framework and methodology

Principle. The framework is hosted on the OpenAlea platform (Pradal *et al.*, 2008). This platform already provides a number of functionalities that match the concepts presented above.

First, the OpenAlea platform operates with a component-based software strategy that facilitates the integration of heterogeneous models into comprehensive assemblies (Pradal *et al.*, 2008). This strategy is centred on the high-level Python language and on scientific workflows. The object-orientated and interpretive Python language has powerful gluing capabilities to integrate existing computational methods written in various languages (Fortran, C, C++) as software components (Pérez et al., 2011) rather than stand-alone programs.

Scientific workflows promote deconstruction of complex models into independent submodels or components that can be recombined dynamically (Gil *et al.*, 2007). These ideas have been extended and specialized to the simulation of plants interacting with their environment to build coherent and modular FSPMs using a collection of models performing elementary tasks (Fournier *et al.*, 2010).

The OpenAlea platform is freely distributed, with different research groups participating and thus enriching the collection of components, and hence promoting the reuse of various knowledge sources. Some components estimate the structural development of the plant (Boudon *et al.*, 2012), whereas other submodels of microclimate calculate leaf temperature or radiation (Chelle and Andrieu, 1998; Sinoquet *et al.*, 2001).

The platform proposes a generic and indirect mode of communication between components. Communication is achieved solely through a generic and multiscale data-structure, the multiscale tree graph (MTG: Godin and Caraglio, 1998). This indirect communication ensures the modular design of the software architecture because one component can replace another one if it generates the same output (e.g. temperature computation on leaves). The MTG represents both the topological functional network of organs and the geometrical arrangement of organs in the same structure.

Furthermore, control of the simulation is delegated to the scientific workflow. This can be used to combine different submodels running at different time steps and investigate different scheduling strategies without changing the submodels themselves.

Finally, in the visual programming environment VisuAlea, the modeller can inspect the structure of the model, run it and explore its outputs in a graphical environment.

Overall, the platform OpenAlea provides a standardized way to build models representing multiple processes. Models may operate at different time steps during a simulation and on different parts of a well-designed shared object representing the canopy (MTG). The platform also offers a collection of plant and physical models that are readily usable to model pathosystems.

Extension to modelling of pathosystems. General OpenAlea methods were extended to the modelling of pathosystems. First, the framework requires canopy models to be wrapped as OpenAlea components compatible with the MTG structure, i.e. it must be able to take one MTG as input and provide one as output. Note that MTGs have various constructors allowing the plant objects to be parameterized as tables of data or as L-system axial trees.

The MTG ensures modular communication between models with various time steps and at different spatial scales (Fig. 3). This central data structure is available at any time for all models. Each model reads and updates information on specific regions of the data structure at its own pace.

The phyto-element should be chosen among MTG entities. As presented in the conceptual model, this entity should have a geometric representation, to be compatible with environmental models, and should be properly connected to plant topology to be compatible with functional models. Basic fungal models may require calculations of leaf surface or senescence.

In a general perspective, every model in charge of a physical or biological process in the pathosystem must fulfil the two conditions mentioned above: availability on OpenAlea and compatibility with MTGs. Note that various models for microclimate, dispersal and plant functioning are available on the OpenAlea platform and accessible by browsing.

For modelling foliar funguses, we provide generic data structures representing the lesions and the dispersal units, and define generic protocols that manage the communication with the MTG. The fungal data structures are designed with an object-orientated approach. Abstract virtual classes with generic interfaces indicate how to model a lesion and a dispersal unit. The

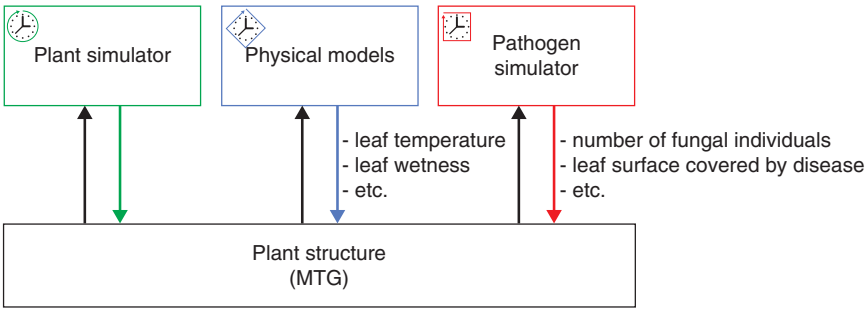

FIG. 3. Methodology implemented for the communication between pathogen–FSPM–physical models through a central data structure representing the canopy. Arrows: flows of information. The different clocks indicate different temporal scales of computation for the components of the pathosystem managed with different rhythms of access to the MTG structure.

modeller encodes methods to calculate processes identified in the conceptual model (e.g. individual growth, ageing and production of spores for the lesions, infection for the dispersal unit). Abstract interfaces leave great latitude to specialize algorithms representing a specific disease within the classes. For example, infection by the dispersal unit can depend only on temperature or be more complex if leaf wetness is involved. Besides, new models can be largely inspired by existing ones because several fungi can share common behaviours.

For the processes that were identified as external models (e.g. dispersal, washing), generic application programming interfaces (APIs) were defined to specify how models should interact with each other and with components of the pathosystem, and a first collection of models is proposed. They can be transposed to different funguses because they are simple enough and no knowledge on the internal functioning of fungal individuals is needed. By contrast, new models can also be very specific to a given species and may require new code to be written.

The protocols of communication between fungus and MTG cover all the interactions identified in the conceptual model (e.g. development of lesions in a population, infection, emission of spores, competition between lesions, transport, response to senescence). These protocols manage the low-level operation with the MTG (reading, writing and traversal), especially the interaction with phyto-elements. The user of the framework can thus concentrate on the actual modelling of processes.

Template dataflow. Tools are provided in OpenAlea to orchestrate calls to asynchronous models. This is illustrated in Fig. 4 with a template dataflow simulating the attack of a fungal disease on an FSPM. The simulation loop is driven by a sequence of discrete events that can be repeated until the end of the simulation. For each step of the simulation, the dataflow is completely evaluated. At initiation, an axiom MTG is generated and inoculated with dispersal units. An axiom MTG is the given plant architecture representation before the first step of the simulation.

In this example, functional nodes in the simulation loop represent models for the '*plant_simulator*', the '*microclimate*', the '*pathogen_simulator*', the '*dispersal*' and the '*outputs_display*'. The order of the nodes is fixed explicitly before the simulation.

The plant structure enters the simulation loop and will circulate through the links between the nodes for each step of the simulation. Therefore, the MTG is updated with a set of expected properties from one node to another. The functional nodes may have several implementations as long as they read and write the same information on the MTG.

Using the information returned by scheduling nodes (blue box), the dataflow orchestrates the calls of functional nodes at variable frequencies. In this particular example, the plant simulator is called every 20°d (Fig. 4-1). Microclimatic variables are updated with an hourly time step, and so is the pathogen simulator (Fig. 4-2). Physical models of dispersal are called only when needed on rain occurrences (Fig. 4-3). Finally, the loop breaks when the first functional node stops.

RESULTS

Integration of two different models of pathosystems

Two existing plant–pathogen models were adapted to our framework: the model Septo3-D of septoria leaf blotch of Robert *et al.* (2008), and the model VignOid of grapevine powdery mildew of Calonnec *et al.* (2008). Both simulate polycyclic foliar funguses on FSPMs and fall within the scope of our study. Both models simulate the invasion of leaves by the fungus. Epidemic cycles have a stepwise structure comprising infection, latency, sporulation and dispersal. The advancement of these stages is calculated deterministically by accumulation of favourable conditions. In addition, in both models, disease dissemination is modulated by canopy architecture. Both plant models use a daily time step, and both fungal models use an hourly time step.

Nevertheless, these pathosystems are different in many regards. Their specificities are discussed here (Table 1). The challenge was to fit these specificities in our generic approach. More than demonstrating a practical application of our framework, focusing on these contrasted pathosystems proves its adaptability.

The equations of the original authors are not changed to any great degree.

Model of septoria leaf blotch

Specificities in wheat–septoria leaf blotch interactions. Wheat is annual and monocotyledonous. Senescence will occur before harvest and it displays heterogeneous tissues within the same leaf starting from the top. *Mycosphaerella graminicola* is hemibiotrophic, so senescence patterns on leaves are expected to influence epidemics. To simulate this, the plant model must simulate intra-leaf heterogeneities.

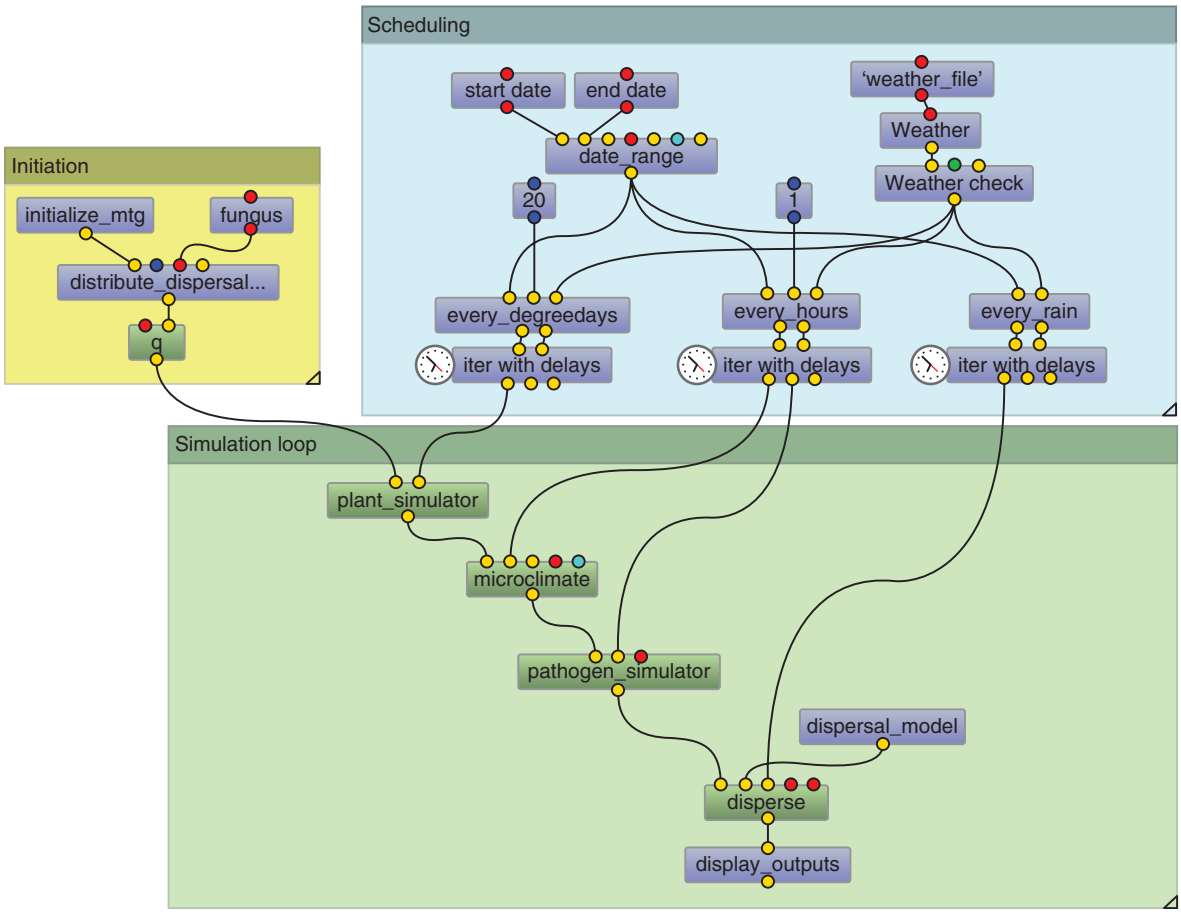

FIG. 4. Example of dataflow simulating an epidemic of septoria on wheat. Frames distinguish nodes of initiation (yellow), simulation loop (green) and scheduling (blue). Links between nodes indicate exchanges of information; the MTG symbolized as '*g*' is updated between each green node. At the beginning of the simulation, the modeller sets the nodes of the scheduler to call models in the loop with a regular frequency in degree-days (here '*plant_simulator*' every 20°d), in actual time (here '*pathogen_simulator*' every 1 h), or only on discrete occurrence in weather data (here '*disperse*' every rain). Different colours in the bullets indicate different types of input. For numbers refer to the text.

TABLE 1. *Comparison of the specificities of the two pathosystems modelled in our framework*

		Pathosytem	
Host		**Wheat**	**Grapevine**
	Classification	Monocotyledon	Dicotyledon
	Life span	Annual	Perennial
	Phyto-element	Leaf region	Leaf
	Senescence accounted	Yes	No
Pathogen		**Septoria leaf blotch**	**Powdery mildew**
	Classification	Ascomycota – Dothideales	Ascomycota – Erysiphales
	Trophic behaviour	Hemi-biotrophic	Biotrophic
	Age-related lesion structure	Yes	No
	Dispersal agent	Rain	Wind

Each lesion of septoria leaf blotch displays an age-related structure, i.e. the fungal tissues in the centre of the lesion are visibly older and in a more advanced stage than the tissues on the periphery (Fig. 5).

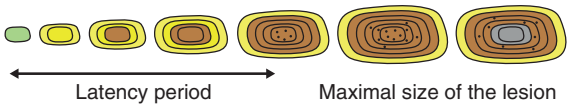

FIG. 5. Schematic representation of the growth of a lesion of septoria leaf blotch.

Finally, *Septoria tritici* is dispersed upward by rainsplash. Its spread is determined strongly by the vertical distances between leaves and by the dynamics of growth of the canopy.

Wheat model. The wheat canopy is simulated with a dynamic architectural model based on the model ADEL (Fournier *et al.*, 2003; Evers *et al.*, 2005), which simulates in three dimensions the dynamics of appearance and growth of all vegetative organs of the wheat canopy. In this model plant development is only a function of the temperature of meristems, approximated here by air temperature above the canopy. One phyto-element is a segment of a leaf, typically 2–3 cm long. Leaf senescence is simulated continuously on the entire leaf. We simplify the physiology of tissues during senescence with binary information:

'*green*' or '*dead*'. However, this notion could be refined with gradients of nitrogen for example.

Lesion. Equation (1) links the growth demand of a lesion ΔS_{lesion} (in cm^2) to the advancement of its age ΔA_{lesion} in degree-days. Degree-days are calculated with leaf temperature approximated from temperature above the canopy. For now the growth rate r is the same for all lesions (Table 2). It was kept the same as in the previous model. Further developments will test variability in growth rates in the population.

$$\Delta S_{\text{lesion}} = r\Delta A_{\text{lesion}} \quad (1)$$

To simulate the age-related structure (Fig. 5), surfaces of the lesion are sorted in different stages (Fig. 6). Surfaces in each stage are distributed in classes of different physiological age. Here the width of classes in degree-days is set to W_{class} to have a smooth enough representation (Table 2). Surfaces pass from one class to the other with ageing as a function of the delta in local thermal time ΔD_{day} (eqn 2). The surface ΔS_{class} passing from one class to another is:

$$\Delta S_{\text{class}} = S_{\text{class}} \frac{\mathbf{\Delta} D_{\mathbf{day}}}{W_{\text{class}}} \quad (2)$$

The developmental steps were refined as shown on Fig. 6. A short necrotrophic stage was added to delay sporulation. The transitions are still deterministic and the thresholds were adapted from the previous model to the new cycle (Table 2).

The first growth ring is the only one undergoing the very first stage of latency. It has been managed separately from the others as an independent object attached to the lesion. Its functioning is close to a lesion of powdery mildew, as detailed in the following section.

Finally, the number of spores produced in a time step is a function of the surface that enters in sporulation, $S_{\text{sporulating}}(t)$. The production rate p was adapted so the stock would be emptied in three dispersal events as in Septo3-D. The stock of spores $Q(t)$ is directly available for dispersal. With $Q(0)$ null before any surface enters sporulation, the accumulation of spores in the stock is calculated as follows:

$$Q(t+1) = Q(t) + pS_{\text{sporulating}}(t) \quad (3)$$

A senescence response has been integrated in the lesion. If senescence occurs during the lifetime of a lesion, it kills all the surfaces under the necrotrophic stage. Practically, it empties classes with surfaces under necrosis. The other surfaces of the lesion remain unaffected.

External models for the lesion. In this first integration of the model of septoria leaf blotch, the simplest strategy of competition for space was used. The green area available on the leaf is homogeneously distributed between lesions, regardless of their developmental stage or position. However, our framework can be extended to manage rules of priority as in the previous model.

Concerning the senescence model, lesions are positioned by their centre on the axis of the leaf, and so does the senescence.

TABLE 2. *List of parameters for the model of septoria leaf blotch (Robert* et al.*, 2008)*

Parameter (unit)	Symbol	Value		Eqn
Lesion				
Growth rate ($cm^2\ °Cd^{-1}$)	r	$1.36e^{-4}$	if age lesion $< A_{chlorosis}$	1
		$6.0e^{-4}$	if age lesion $< A_{chlorosis}$ and surface lesion $< S_{\max}$	
		0	otherwise	
Age threshold for chlorosis (°Cd)	$A_{\text{chlorosis}}$	220		
Age threshold for necrosis (°Cd)	$A_{necrosis}$	20		
Age threshold for sporulation (°Cd)	$A_{\text{sporulation}}$	110		
Width of age classes (°Cd)	W_{class}	20		2
Maximal surface (cm^2)	$S_{\max}$	0·3		
Sporulation rate (spores cm^{-2})	p	$1e^5$		3
Dispersal unit				
PAR threshold	$PAR_{\text{threshold}}$	644		
Relative humidity threshold	$RH_{\text{threshold}}$	85		
Hourly lost rate		0·008		

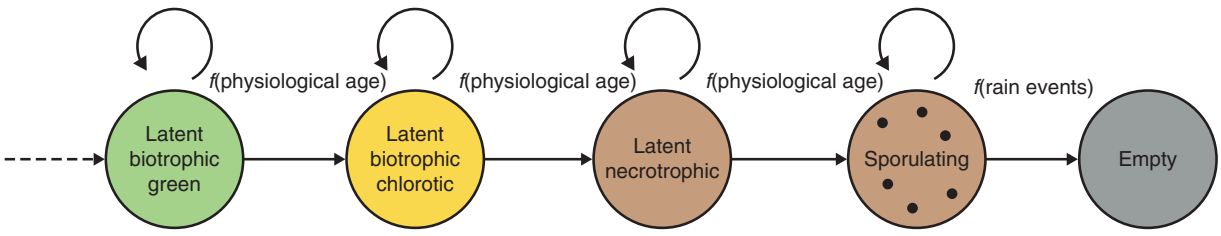

FIG. 6. Developmental stages of surfaces on a lesion of septoria leaf blotch.

If senescence reaches them, their method of senescence response is called.

Dispersal unit. For each dispersal unit, the infection is completed after 10 h of high humidity on the phyto-element: PAR $<$ $PAR_{threshold}$ and relative humidity $>$ $RH_{threshold}$. Until infection succeeds, an hourly lost rate of dispersal units has been set (Table 2).

External models for the dispersal unit. Dispersal units are positioned on the leaf axis. Infection can only occur on green healthy tissues. The infection model compares their position to senescence progress. If they are not on senesced tissues, the infection model compares the surface of lesions not yet reached by senescence and the surface of non-senescent tissues to eliminate dispersal units in a probabilistic approach.

Furthermore, if another rainy event occurs, the probability of the dispersal unit being washed off the leaf is calculated with a physical model adapted from Rapilly and Jolivet (1976).

Dispersal. The model of dispersal of Septo3-D was simply encapsulated in the new framework, so the computation stays the same.

The number of infectious droplets emitted is computed in layers in one dimension. It depends on rain intensity reaching these layers and on the sporulating area in these layers. The choice of layer height was reasoned by Robert *et al.* (2008) to be 1 cm, i.e. approximately five times shorter than the size of a phyto-element.

Transport is also calculated in one dimension. Dispersal units travel a limited distance upward, in the hemisphere perpendicular to the surface source leaf. The density of emitted droplets decreases exponentially with distance. They then fall vertically with gravity. During both movements, droplets are intercepted or not by the vegetation in layers.

Illustration with simulation outputs. The Figure 7 follows the fate of dispersal units on the top leaf of a wheat plant during one simulation. After deposit on this leaf, several dispersal units might be washed off and the remaining units can only infect healthy tissues. Despite the large number of deposits, only a few will actually achieve infection.

Figure 8 shows a 3-D visualization of the epidemics. Severity is calculated on each leaf, and is the ratio between the disease surface and leaf surface.

Model of grapevine powdery mildew

Specificities in grapevine-powdery mildew interactions. Grapevine is perennial and dicotyledonous. Senescence will not occur before harvest, and hence it will not be taken into account in this case. The ageing of one leaf is homogeneous on its entire surface. In this case, no infra-leaf variability will be simulated on grapevine tissues. Powdery mildew is strictly biotrophic and it is influenced by the age of leaves. No age-related structure on lesions has been pointed out for this fungus. All tissues of the same lesion are the same age. Powdery mildew is dispersed by wind. It is influenced by the density and the vigour of the canopy.

Grapevine. The architecture of the grapevine stock is captured in a dynamic 3-D Lsystem based on TopVine (Louarn *et al.*, 2008). L-Py, an LSystem plant simulation program, is used to simulate grapevine growth (Boudon *et al.*, 2012). The output of each simulation step is an MTG. Phyto-elements are entire leaves with attached properties such as surface, age, position and geometry.

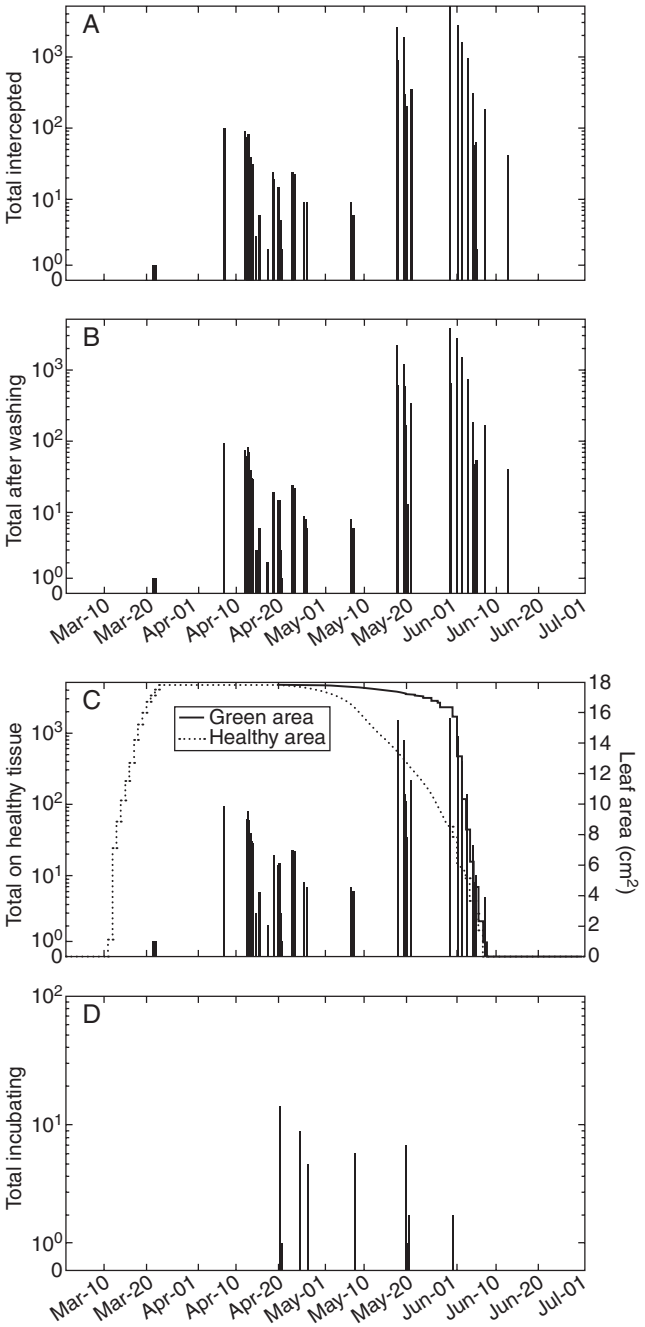

FIG. 7. Example of simulation plotted over time (weather: Grignon, 2001). (A) Number of dispersal units received by top leaf. (B) Number of dispersal units remaining after washing. (C) Number of dispersal units hitting healthy tissues. Curves indicate green leaf area and healthy area (green without lesions). (D) Number of new infections.

Lesion. Laws for growth, ageing and sporulation were kept as in VignOid. Growth, however, was encoded in the new system of growth demand and growth offer. At each time step, growth demand is obtained after a diameter increase demand ΔD_{lesion}.

It is calculated as in Calonnec *et al.* (2008), with K_{max} the maximum colony diameter, r the growth rate, t^* the time to $K_{max}/2$, and $F(T_n)$ the temperature factor.

$$\Delta D_{lesion} = K_{max} F(T_n) \frac{r e^{r(t^*-t)}}{[1 + e^{r(t^*-t)}]^2} \quad (4)$$

with $F(T_n)$ a function of a normalized temperature T_n:

$$F(T_n) = \frac{(m+n)^{(m+n)}}{n^n m^m} \text{ and } T_n = \frac{T(t) - T_{min}}{T_{max} - T_{min}} \quad (5)$$

where $T(t)$ is the temperature at time t and T_{min} and T_{max} are cardinal temperatures, with m and n shape parameters of the curve. The parameters given in the previous model were used.

FIG. 8. Example of visualization of simulation outputs of septoria leaf blotch on wheat. Severity is displayed with a gradual colourmap on leaves. Brown leaves on the bottom are fully senesced. A video showing a dynamic 3-D visualization of simulation outputs of septoria leaf blotch on wheat is available as Supplementary Data.

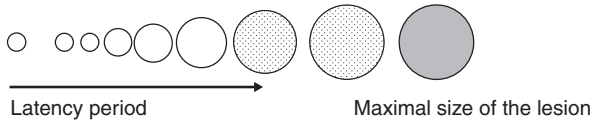

FIG. 9. Schematic representation of the growth of a lesion of powdery mildew.

The value of K_{max} is a function of leaf age (in days), where A, B and c are scaling parameters, and age_{leaf} is measured in days:

$$K_{max} = (A - B)e^{(-c\, age_{leaf})} + B \quad (6)$$

The transitions between developmental stages depend on the cumulative temperature. All the surface of a lesion is the same stage (Figs 9 and 10).

With ρ_{min} the minimum time before the appearance of first spores, the latent period is completed when:

$$\sum \frac{F(T_n)}{\rho_{min}} = 1 \quad (7)$$

The quantity of spores produced hourly by a lesion is a function of its surface S_{lesion}, where β and δ are scaling parameters:

$$Q = \beta e^{(\delta S_{lesion})} \quad (8)$$

The progress in the period of sporulation is calculated as for the progress in latency period but with different parameters.

Dispersal. The fraction of spores released (Q_r) was described by Willocquet *et al.* (1998) and depends on the wind speed on the phyto-element u (in m s^{-1}) approximated from wind data above the canopy:

$$Q_r = Q \frac{e^{(\tau u + b)}}{1 + e^{(\tau u + b)}} \quad (9)$$

where τ is the relative increase of dispersed spores per unit of wind speed increase, relative to the fraction of spores available for dispersal, and a and b are parameters. In this model, a single spore counts as a dispersal unit.

The dispersal units are distributed on target leaves in a 3-D cone of dispersal in the direction of the wind. The chances of a dispersal unit reaching a leaf decrease exponentially as the distance d (in cm) from the source increases (c_{id} is the decay rate with distance), and also as the angle θ (in degrees) of the source–target leaf trajectory increases with respect to the wind direction. However, the fraction of spores reaching a leaf, Q_{leaf}, increases with target leaf surface S_{leaf} (in cm^2):

$$\begin{aligned} Q_{leaf} &= Q_r S_{leaf} e^{(-c_{id} d)} r(\theta) \text{ with} \\ r(\theta) &= \frac{1}{10_0}(\alpha_0 - \theta)/\alpha_0 \end{aligned} \quad (10)$$

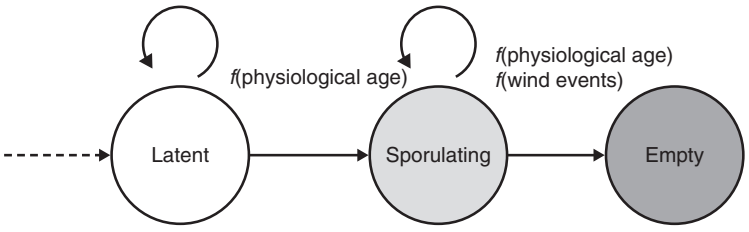

FIG. 10. Developmental stages of a lesion of powdery mildew.

Dispersal unit. The calculation of infection of the previous model was adapted to an hourly time step. Each dispersal units accumulates a physiological age in degree-days. According to the literature, the assumption was made that a certain age $A_{\text{threshold}}$ was required to perform infection (Table 3).

After this time, the success of infection I is a function of temperature T (in °C) and leaf age age_{leaf} as in the previous model:

$$I = I_0 F(T_n) e^{(-\tau age_{\text{leaf}})} \quad (11)$$

TABLE 3. *List of parameters for the model of powdery mildew (Calonnec et al., 2008)*

Parameter (unit)	Symbol	Value	Equation
Lesion growth			
Growth rate	r	0·2	4
Time at 50% of growth (d)	t^*	13	4
Shape parameter	$n - m$	1·24–0·27	5
Cardinal temperatures (°C)	$T_{\max} - T_{\min}$	33–5	5
Maximum, minimum diameter (mm)	A–B	18–2	6
Rate of colony growth with the leaves age	c	0·08	6
Lesion latent period			
Minimum time for latency (d)	$\rho_{\min}$	6	7
Lesion sporulation			
Length of sporulation period (d)	L_{spo}	10	
Parameters in the spore production function	β–δ	36–0·314	8
Dispersal			
Parameter in the spore release function	τ–b	0·71–5·8	9
Spore decay with distance	c_{id}	0·04	10
Angle of the cone relative to wind direction (°)	α_0	45	10
Dispersal init infection			
Maximum infection rate	I_0	0·53	11
Leaf susceptibility decay with age	τ	0·147	11
Shape parameter	n–m	1·055–0·338	11
Cardinal temperatures (°C)	$T_{\max} - T_{\min}$	33–5	11

where $F(T_n)$ is the function of normalized temperature, I_0 is the maximum infection rate at optimum temperature and τ is the decay rate of leaf susceptibility.

External infectious and growth models. The infection and growth models are the same as in the model of septoria leaf blotch.

Illustration with simulation outputs. Figure 11 shows the coverage of a single grapevine leaf by the disease during an example simulation. At the beginning, several lesions appear in a latent state. The lesions then sporulate progressively. After several dispersal events and if their sporulating period is over, the lesions are empty.

Figure 12 shows a visualization of the epidemics in three dimensions. In this case the severity is also calculated on each leaf.

Modularity of the framework: comparison of two models of dispersal in three dimensions

The modularity of the framework was tested in the following application. Two varied models of dispersal are exchanged: we simulate the distribution of the dispersal units on different leaves of the canopy for two architectures when the dispersal is caused by rain-splash or by wind

Materials and methods. Two canopies of wheat are generated at 1500°d with ADEL wheat. Density is set to 250 plants m−2 and we simulate a plot of 1 × 1 m. The two canopies are generated with mock-ups of wheat from two varieties ('Cap Horn' and 'Soissons') that differ in leaf area index (LAI) and height (Fig. 13).

Two models of dispersal are compared for one dispersal event from a single source: dispersal by rain and dispersal by wind. The source leaf, in the centre of the canopy, bears one lesion emitting 10^4 dispersal units only once. The model of dispersal by wind operates as in the powdery mildew model. The model of dispersal by rain used for *Septoria tritici* has been transformed for a use in three rather than one dimension. The principle and the parameters are the same but phyto-elements are treated individually in the hemisphere perpendicular to the surface of the source and

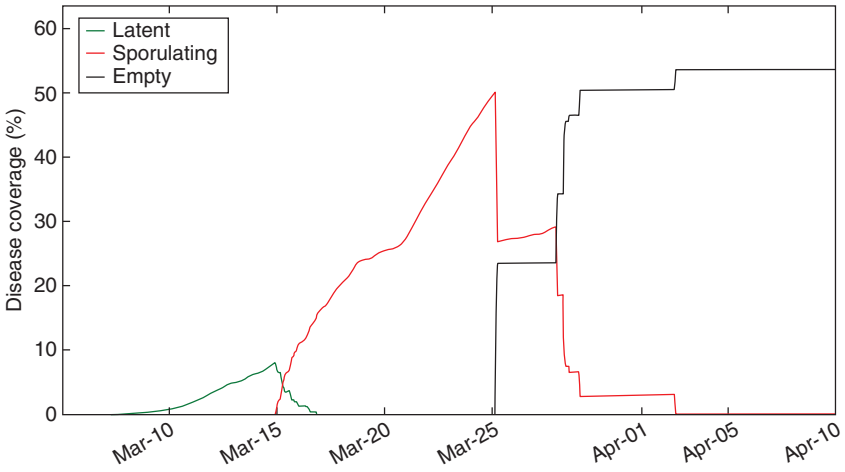

FIG. 11. Example of disease coverage on one leaf during a simulation of powdery mildew plotted over time (for details of weather conditions see Grignon, 2001).

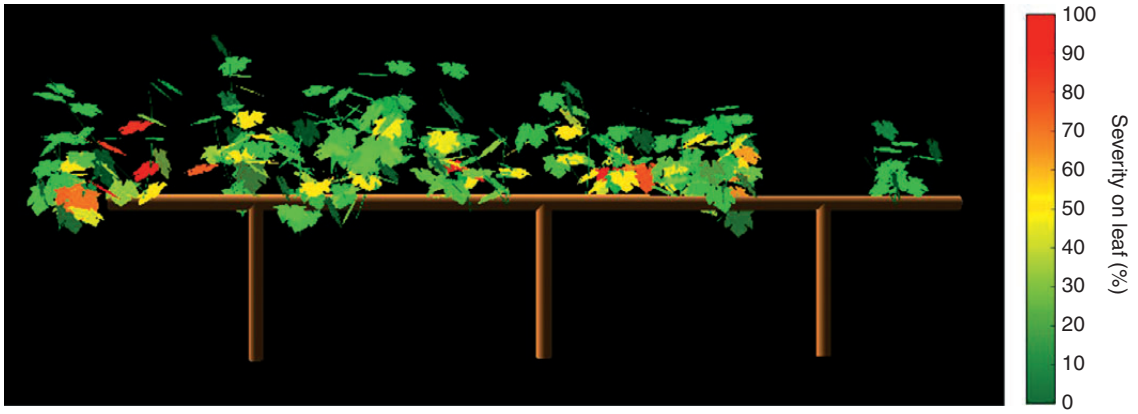

FIG. 12. Example of visualization of simulation outputs of powdery mildew on grapevine. Severity is indicated by the colour coding.

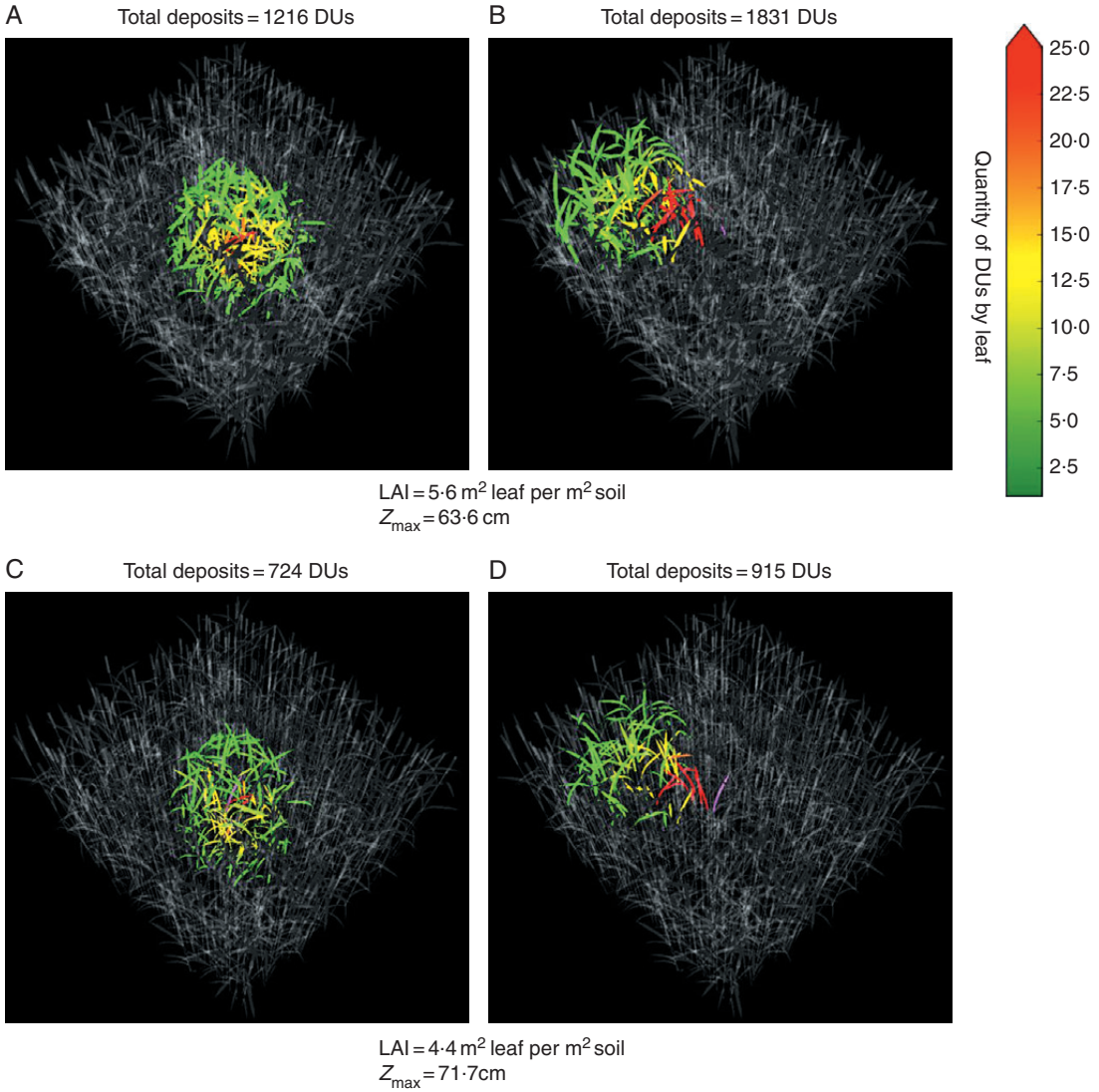

FIG. 13. Distribution of dispersal units after one dispersal event due to rain (A, C) or wind (B, D) (with same source leaf, indicated in pink on the figure) on two wheat cultivars with different architectures (A, B) 'Cap Horn' and (C, D) 'Soisson'. Leaf area index is higher and stem height is lower in architecture 'Cap Horn' compared with 'Soisson' (parameters given in the figure). LAI, leaf area index; Z_{max}, maximal height of canopy; DUs, dispersal units. Note that the wind direction is towards the top left corner of the figure.

not grouped into layers. They are sorted from closest to the source leaf to furthest. In this order, the number of deposits is calculated on each target as a function of its distance from the source and its surface. In a second run, the same method is used to deposit dispersal units downward on phyto-elements in the vertical projection of the hemisphere.

Results. The source leaf is coloured in pink. A gradient of colour from green to red indicates the number of dispersal units deposited on each leaf after the spore dispersal event. Patterns of distribution of dispersal units are different depending on the dispersal type strategy. Rain produces ellipsoidal distributions around the leaf source (Fig. 13A, C) whereas wind allows spores to reach leaves in a conical shape (Fig. 13B, D). Wind is more efficient in terms of number of intercepted dispersal units after one event. Similar effects of wind and rain are observed for the two different wheat architectures. Yet, fewer dispersal units are intercepted by the leaves when the canopy is higher and less dense (Fig. 13A, B and 13C, D).

Modularity of the framework: influence of architectural traits of canopies on dispersal by rain

The modularity of the framework was also tested in the following application. We simulate the distribution of the dispersal units on different leaves of the canopy when dispersal is caused by rain-splash, for 10 stem heights × 7 densities of canopies × 2 heights for the source leaf.

Materials and methods. Canopies of wheat are generated at the age of 1500°d with ADEL wheat. We simulate a plot of 1 × 1 m. We simulate one dispersal event by rain-splash using the model in three dimensions as above. For the first set of simulations, the source of dispersal units is the lowest leaf of a stem in the centre of the canopy. For the second set, the source is positioned on a leaf at 2/3 of stem height. The simulations are run for seven densities staggered from 50 to 350 plants, × 10 stem heights staggered from 20 to 120 cm. In each simulation, we calculate: the total number of dispersal units deposited on leaves, the number of dispersal units deposited on flag leaves and the number of flag leaves touched by at least one dispersal unit.

Results. Comparing Fig. 14A and D shows that the total number of deposits depends strongly on the location of the source of dispersal units: the number of deposits is very low for a source located on the lowest leaf. This is due to the simulated trajectory of a splashed drop from the source, which goes up 20 cm before falling to the ground; a trajectory starting at 2/3 of stem height is therefore much longer and has a much higher probability of encountering leaves. These panels further illustrate that the effect of the two traits of architecture (LAI and height) interact with the source location. Increasing LAI increases the total number of deposits but this effect is much stronger for a higher source location (Fig. 14D). Increasing stem height decreases the total number of deposits but only for a low source location (Fig. 14A). Figure 14B and E show that the number of deposits on flag leaves depends on the distance between the source and the flag leaf. Dispersal from the lower leaf is possible only in dwarf wheat (Fig 14B). In terms of the number of flag leaves reached by dispersal units (Fig 14C, F), the overall pattern is very similar to that in terms of the number of deposits (Fig 14B, E), but there is an important quantitative difference: the difference is smaller in terms of the number of leaves touched. This implies that a low source location produces not only fewer leaves touched by dispersal units but also fewer dispersal units per leaf, both of which can influence epidemics.

DISCUSSION

We have developed a modelling framework for the dynamics of foliar fungal pathosystems with explicit coupling of plant architecture, foliar fungal pathogens and microclimate. Components and interactions are based on biological knowledge of a wide range of foliar pathosystems (see 'The foliar fungal pathosystem'). The framework provides an implementation of these concepts in the field of structural–functional models. It should help to understand and quantify effects of plant architecture, on epidemics.

Interactions are addressed at narrow scales. The canopy is decomposed in scales ranging from the square centimetre of leaf tissue up to a plant population of a few square metres.

The cm^2 resolution enables us to account for local leaf physiology and foliar microclimate, required for modelling the infection cycle. The m^2 scale enables us to model short- and medium-range spore dispersal depending on canopy architecture. The concept of a phyto-element plays a central role in the breakdown of our system, as it defines the scale of interactions between pathogens and microclimate. Its choice has to be consistent with the precision of bio-physical models, and is very much linked to the scientific progress in this area. For light models, precision to the nearest centimetre is achievable (Chelle, 2005), but it requires a valid 3-D structure. For wind and temperature models, the question remains open, although recent works are encouraging (Tuzet and Wilson, 2002; Saudreau *et al.*, 2007). Models of leaf wetness are currently less detailed, although they would be greatly beneficial to disease modelling at the leaf scale. Further developments of our framework should investigate the applicability of simulated epidemics to the size of phyto-elements.

The fungal population is decomposed into lesions and dispersal units. Our individual-based approach helps phytopathologists to integrate knowledge from a biological point of view because they relate more easily to observations of fungal behaviour. This could lead to improved modelling of the local response of dispersal units and lesions to their environment. Robert *et al.* (2008) acknowledged the difficulty of integrating fungal reactions to senescence in their epidemic model. In our framework, knowing the localization of the individual lesions on the leaf facilitates modelling the effect of senescence and competition between lesions.

The performance of our approach was tested in a benchmark (Fig. 15). Our approach is suited for canopies of a few square metres. The simulation time of an epidemic takes from less than 1 min for one plant to more than 1 h for 20 plants. To simulate complex epidemics at a higher scale, multi-scale models such as that proposed by Mammeri *et al.* (2014) are needed. Plant–pathogen interactions are simulated down to the organ scale using an FSPM, like the one presented here, and the output is used to calibrate the parameters of a continuous model at the plot scale.

FIG. 14. Effects of LAI and stem height on splash dispersal, assuming a single source of spores (104 dispersal units). Height (cm) indicates the distance from ground to the flag leaf ligule. LAI indicates the leaf area index of the canopy. Upper panels: the source of dispersal units is located on the lowest leaf. Lower panels: the source of dispersal units is located at 2/3 of the stem height. (A, D) Total number of dispersal units deposited on leaves. (B, E) Number of dispersal units deposited on the flag leaves. (C, F) Number of flag leaves touched by at least one dispersal unit.

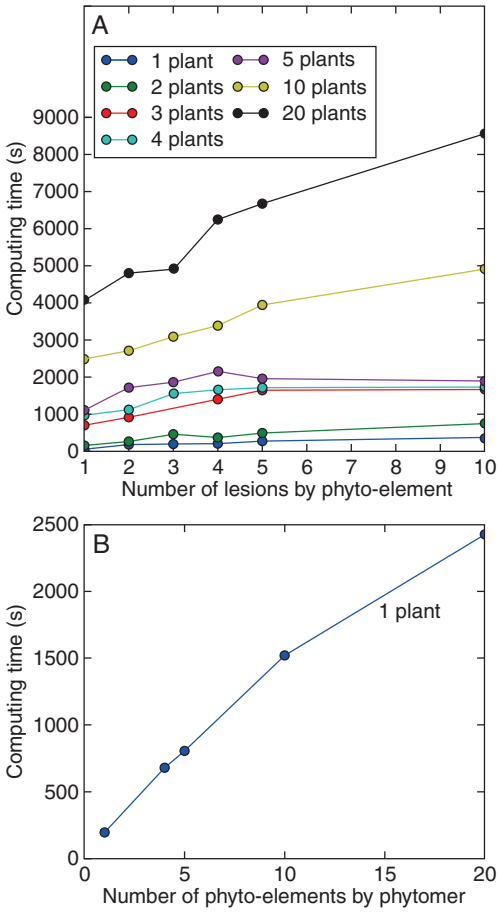

FIG. 15. Performance of a simulation of septoria leaf blotch on wheat. All benchmarks are executed during 1000°d, with a rain event every 50°d and optimal conditions for fungal development. At its final stage of development, each wheat plant contains 56 phytomers. The benchmarks are executed for different numbers of plants, phyto-elements (i.e. leaf sectors) and lesions on each phyto-element at the initial time point. The benchmarks are executed on a laptop computer MacBook Pro equipped with Intel Core2 i7 CPU@2·3 GHz with 8 GB of memory and Python 2·7.6. (A) Total time as a function of the initial number of lesions by phyto-elements. (B) Total time as a function of the number of phyto-elements by phytomer. All benchmarks are executed only once with a fixed seed. Total time varies from 1 min for one plant to 1 h for 20 plants. This time increases linearly with the number of botanical entities N in the MTG with $N =$ (no. of plants) (no. of phytomers) (no. of phyto-elements).

We argue that our framework captures the key interactions for highlighting architectural traits that influence diseases. Furthermore, the resolution we use is appropriate to accommodate tactical operations: for example, models of pruning, shoot topping or phytosanitary treatments can be coupled. Strategic decisions such as choice of variety can also be tested with our approach.

The applicability of the framework was tested by applying it to two contrasted pathosystems: *Mycosphaerella graminicola* on wheat and *Uncinula necator* on grapevine, which differ in terms of spore dispersal, plant architecture and type of infection cycle. We implemented the two systems and simulated the two types of epidemics. This was made possible by the modular structure of the framework, coupling the models of (1) short- and medium-range spore dispersal; (2) the infection cycle; (3) plant architecture; and (4) physical models for the microclimate. The methodology we used enables managing different time scales between submodels. Submodels communicate through the MTG on which they read and write information at their own pace. This structure enables one to modify one or more submodel without changing the rest. Each submodel can be recycled for different pathosystems. We used different models of dispersal and of lesion development without altering the operation of the system. In a caricatured test of this particular feature, we simulated a powdery mildew epidemic on wheat.

Using our modelling framework should facilitate the development of models for new pathosystems by: (1) developing and assembling new modules in the framework (e.g. a new infection cycle module); or (2) making new combinations using modules currently available (e.g. different formalisms for wind-related spore dispersal). Programming skills are required for new developments and, depending on the complexity, the time required will vary from a few hours to several months.

The framework is suited for the integration of experimental results with simultaneous characterization of host, fungus and/or microclimate. For instance, recent findings on septoria leaf blotch encourage us to update our parameterization of the response of lesions to temperature (Bernard *et al.*, 2013).

We hope that the integration of this framework and its free distribution in the collaborative platform OpenAlea will stimulate the emergence of a library of fungus models compatible with FSPM models available on the platform. The library already contains the models of dispersal units, lesions and dispersal strategies presented in this study. Algorithms can be picked up and extended for the construction of future models. As a result, disease modelling could benefit from an active and multidisciplinary scientific community.

Tools are also provided for the analysis of the pathosystem during simulations. The platform readily allows visualization of plants in three dimensions with its infected leaves coloured as a function of disease severity. Quantities of dispersal units can be followed at different locations of the canopy. Severity curves can also be drawn for each phyto-element, or aggregated at the leaf, plant or canopy level. Sensitivity and uncertainty analyses can be run on individual components of the system or on system subsets.

One line of future research concerns modelling of the interactions between the pathogen and the physiological status of leaves. This feature has been anticipated in our framework but the incorporation of leaf functioning in FSPM has just started (Bertheloot *et al.*, 2011) and no FSPMs that describe physiological leaf status are currently available. A second line of future research concerns the effects of the disease on plant functioning. Fungal lesions reduce the photosynthetic activity of the leaves and induce perturbations in foliar tissues (Moriondo *et al.*, 2005; Robert *et al.*, 2006). Third, in crops, fungal species can coexist in the same host and compete on the same leaves (Robert *et al.*, 2004). It should be possible to model such foliar fungal complexes in our framework due to its modular and object-orientated structure and the possibility of modelling different types of infection cycles and spore dispersal. Finally, it would be valuable to extend the framework by including other types of plant–fungus interfaces such as stems, flowers and fruits that are represented in FSPMs. The definition of a phyto-element would have to be extended. This would also

require the integration of new types of communication at these interfaces in the framework.

SUPPLEMENTARY DATA

Supplementary data are available online at www.aob.oxfordjournals.org and consist of a video showing the dynamic 3-D visualization of simulation outputs of septoria leaf blotch on wheat.

ACKNOWLEDGEMENTS

G.G. benefited from a grant by ANRT (CIFRE no. 2012/0406). This work was supported by an Agropolis Fondation "OpenAlea" Grant. We thank Michael Mielewczik for improving the English text.

LITERATURE CITED

Ando K, Grumet R, Terpstra K, Kelly JD. 2005. Manipulation of plant architecture to enhance crop disease control. *CAB Reviews: Perspectives in Agriculture, Veterinary Science, Nutrition and Natural Sources* 2.

Aubertot JN, Barbier JM, Carpentier A, *et al*. 2007. *Pesticides, agriculture et environnement: Réduire l'utilisation des pesticides et limiter leurs impacts environnementaux*. Versailles: Editions Quae.

Austin CN, Wilcox WF. 2012. Effects of sunlight exposure on grapevine powdery mildew development. *Phytopathology* **102**: 857–866.

Baccar R, Fournier C, Dornbusch T, Andrieu B, Gouache D, Robert C. 2011. Modelling the effect of wheat canopy architecture as affected by sowing density on *Septoria tritici* epidemics using a coupled epidemic-virtual plant model. *Annals of Botany* **108**: 1179–1194.

Bahat A. 1980. Factors affecting the vertical progression of septoria leaf blotch in short-statured wheats. *Phytopathology* **70**: 179.

Barthélémy D, Caraglio Y. 2007. Plant architecture: a dynamic, multilevel and comprehensive approach to plant form, structure and ontogeny. *Annals of Botany* **99**: 375–407.

Bassette C, Bussière F. 2005. 3-D modelling of the banana architecture for simulation of rainfall interception parameters. *Agricultural and Forest Meteorology* **129**: 95–100.

Bernard F, Sache I, Suffert F, Chelle M. 2013. The development of a foliar fungal pathogen does react to leaf temperature! *The New phytologist* **198**: 232–240.

Bertheloot J, Cournède P-H, Andrieu B. 2011. NEMA, a functional–structural model of nitrogen economy within wheat culms after flowering. *I. Model description.* Annals of Botany **108**: 1085–1096.

Bolton MD, Kolmer JA, Garvin DF. 2008. Wheat leaf rust caused by *Puccinia triticina*. *Molecular Plant Pathology* **9**: 563–575.

Boudon F, Pradal C, Cokelaer T, Prusinkiewicz P, Godin C. 2012. L-py: an L-system simulation framework for modeling plant architecture development based on a dynamic language. *Frontiers in Plant Science* **3**: 76.

Burie JB, Langlais M, Calonnec A. 2011. Switching from a mechanistic model to a continuous model to study at different scales the effect of vine growth on the dynamic of a powdery mildew epidemic. *Annals of Botany* **107**: 885–895.

Calonnec A, Cartolaro P, Naulin J-M, Bailey D, Langlais M. 2008. A host–pathogen simulation model: powdery mildew of grapevine. *Plant Pathology* **57**: 493–508.

Calonnec A, Burie JB, Langlais M, *et al*. 2012. Impacts of plant growth and architecture on pathogen processes and their consequences for epidemic behaviour. *European Journal of Plant Pathology* **135**: 1–19.

Carbonneau A, Cargnello G, Groupe d'étude des Systèmes de Conduite de la Vigne. 2003. *Architectures de la vigne et systèmes de conduite*. Paris: Dunod: Éditions La Vinge.

Casadebaig P, Quesnel G, Langlais M, Faivre R. 2012. A generic model to simulate air-borne diseases as a function of crop architecture. *PLoS ONE* **7**: e49406.

Caubel J, Launay M, Lannou C, Brisson N. 2012. Generic response functions to simulate climate-based processes in models for the development of air-borne fungal crop pathogens. *Ecological Modelling* **242**: 92–104.

Chelle M. 2005. Phylloclimate or the climate perceived by individual plant organs: what is it? How to model it? What for? *The New Phytologist* **166**: 781–790.

Chelle M, Andrieu B. 1998. The nested radiosity model for the distribution of light within plant canopies. *Ecological Modelling* **111**: 75–91.

Chelle M, Gutschick V. 2010. A nested approach to model the temperature of individual leaves within crop canopies. In: DeJong T, Da Silva D, eds. *6th International Workshop on Functional-Structural Plant Models*. Davis, USA: University of California, 129–131.

Cieslak M, Seleznyova AN, Prusinkiewicz P, Hanan J. 2011. Towards aspect-oriented functional–structural plant modelling. *Annals of Botany* **108**: 1025–1041.

Costes E, Lauri PE, Simon S, Andrieu B. 2013. Plant architecture, its diversity and manipulation in agronomic conditions, in relation with pest and pathogen attacks. *European Journal of Plant Pathology* **135**: 455–470.

Dauzat J, Rapidel B, Berger A. 2001. Simulation of leaf transpiration and sap flow in virtual plants: model description and application to a coffee plantation in Costa Rica. *Agricultural and Forest Meteorology* **109**: 143–160.

Evers JB, Vos J, Fournier C, Andrieu B, Chelle M, Struik PC. 2005. Towards a generic architectural model of tillering in Gramineae, as exemplified by spring wheat (*Triticum aestivum*). *The New Phytologist* **166**: 801–812.

Eversmeyer MG, Kramer CL. 2000. Epidemiology of wheat leaf and stem rust in the central great plains of the USA. *Annual Review of Phytopathology* **38**: 491–513.

Eyal Z. 1971. The kinetics of pycnidiospore liberation in *Septoria tritici*. *Canadian Journal of Botany* **49**: 1095–1099.

Fournier C, Andrieu B. 1998. A 3-D architectural and process-based model of maize development. *Annals of Botany* **81**: 233–250.

Fournier C, Andrieu B, Ljutovac S, Saint-Jean S. 2003. ADEL-wheat: a 3-D architectural model of wheat development. In: Hu B-G, Jaeger M, eds. *Plant Growth Modeling and Applications, Proceedings of 2003 International Symposium*. Beijing, China: Tsinghua University Press - Springer, 54–63.

Fournier C, Pradal C, Louarn G, Combes D, Soulié J-C, Luquet D, Boudon F, Chelle M 2010. Building modular FSPM under OpenAlea: concepts and applications. In: DeJong T, Da Silva D, eds. *6th International Workshop on Functional-Structural Plant Models*. Davis, USA: University of California, 109–112.

Fournier C, Pradal C, Abichou M, *et al*. 2013. An integrated and modular model for simulating and evaluating how canopy architecture can help reduce fungicide applications. In: Sievänen R, Nikinmaa E, Godin E, Lintunen A, Nygren P, eds. *7th International Conference on Functional-Structural Plant Models*. Saariselkä, Finland, 345–348.

Gadoury DM, Stensvand A, Seem RC. 1998. Influence of light, relative humidity, and maturity of populations on discharge of ascospores of *Venturia inaequalis*. *Phytopathology* **88**: 902–909.

Galet P. 1977. *Les maladies et les parasites de la vigne – Tome 1*. Montpellier: Imprimerie du Paysan du Midi.

Giavitto J-L, Malcolm G, Michel O. 2004. Rewriting systems and the modelling of biological systems. *Comparative and Functional Genomics* **5**: 95–99.

Gigot C, Saint-Jean S, Huber L, *et al*. 2013. Protective effects of a wheat cultivar mixture against splash-dispersed septoria tritici blotch epidemics. *Plant Pathology* **62**: 1011–1019.

Gil Y, Deelman E, Ellisman M, *et al*. 2007. Examining the challenges of scientific workflows. *IEEE COMPUTER VOL* **40**: 24–32.

Godin C, Caraglio Y. 1998. A multiscale model of plant topological structures. *Journal of Theoretical Biology* **191**: 1–46.

Jones HG. 1992. *Plants and microclimate: a quantitative approach to environmental plant physiology*. Cambridge: Cambridge University Press.

Leca A, Saudreau M. 2010. Physical modelling of leaf wetness duration at the leaf scale taking into account leaf wettability. In: DeJong T, Da Silva D, eds. *6th International Workshop on Functional-Structural Plant Models*. Davis, USA: University of California, 117–119.

Louarn G, Lecoeur J, Lebon E. 2008. A three-dimensional statistical reconstruction model of grapevine (*Vitis vinifera*) simulating canopy structure variability within and between cultivar/training system pairs. *Annals of Botany* **101**: 1167–1184.

Lovell DJ, Parker SR, Hunter T, Royle DJ, Coker RR. 1997. Influence of crop growth and structure on the risk of epidemics by *Mycosphaerella graminicola* (*Septoria tritici*) in winter wheat. *Plant Pathology* **46**: 126–138.

Lovell DJ, Parker SR, Hunter T, Welham SJ, Nichols AR. 2004. Position of inoculum in the canopy affects the risk of septoria tritici blotch epidemics in winter wheat. *Plant Pathology* **53**: 11–21.

Lucas M, Laplaze L, Bennett MJ. 2011. Plant systems biology: network matters. *Plant, Cell & Environment* **34**: 535–553.

van Maanen A, Xu X-M. 2003. Modelling plant disease epidemics. *European Journal of Plant Pathology* **109**: 669–682.

Madden LV, Hughes G, van den Bosch F. 2007. *The study of plant disease epidemics*. St. Paul, MN: APS Press.

Magarey RD, Seem RC, Russo JM. 2006. Grape canopy surface wetness: simulation versus visualization and measurement. *Agricultural and Forest Meteorology* **139**: 361–372.

Mammeri Y, Burie JB, Calonnec A, Cokelaer T, Costes E, Langlais M, Pradal C. 2010. Modelling of the airborne dispersal of a pathogen over a structured vegetal cover. In: DeJong T, Da Silva D, eds. *6th International Workshop on Functional-Structural Plant Models*. Davis, USA: University of California, 50–52.

Mammeri Y, Burie JB, Langlais M, Calonnec A. 2014. How changes in the dynamic of crop susceptibility and cultural practices can be used to better control the spread of a fungal pathogen at the plot scale? *Ecological Modelling*. In press.

McDonald MR, Gossen BD, Kora C, Parker M, Boland G. 2013. Using crop canopy modification to manage plant diseases. *European Journal of Plant Pathology* **135**: 581–593.

Mech R, Prusinkiewicz P. 1996. Visual models of plants interacting with their environment. SIGGRAPH '96. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York: , 397–410.

Moriondo M, Orlandini S, Giuntoli A, Bindi M. 2005. The effect of downy and powdery mildew on grapevine (*Vitis vinifera* L.) leaf gas exchange. *Journal of Phytopathology* **153**: 350–357.

Moulia B, Edelin C, Loup C, Jeuffroy M-H. 1999. Architectural analysis of herbaceous crop species: a comparative study of maize (*Zea mays* L.) and garden pea (*Pisum sativum* L.). *Agronomie* **19**: 305–312.

Mwakutuya E, Banniza S. 2010. Influence of temperature and wetness periods on the development of *Stemphylium* blight on lentil. *Plant Disease* **94**: 1219–1224.

Ngao J, Adam B, Charreyron M, Saudreau M. 2013. Spatial and temporal variability of leaf gas exchange and temperature responses of apple trees to drought assessed by a 3-D turbid medium model. In: Sievänen R, Nikinmaa E, Godin E, Lintunen A, Nygren P, eds. *7th International Conference on Functional-Structural Plant Models*. Saariselkä, Finland, 118–120.

Pangga IB, Hanan J, Chakraborty S. 2011. Pathogen dynamics in a crop canopy and their evolution under changing climate. *Plant Pathology* **60**: 70–81.

Parent B, Turc O, Gibon Y, Stitt M, Tardieu F. 2010. Modelling temperature-compensated physiological rates, based on the co-ordination of responses to temperature of developmental processes. *Journal of Experimental Botany* **61**: 2057–2069.

Pérez F, Granger BE, Hunter JD. 2011. Python: an ecosystem for scientific computing. *Computing in Science & Engineering* **13**: 13–21.

Pradal C, Dufour-Kowalski S, Boudon F, Fournier C, Godin C. 2008. OpenAlea: a visual programming and component-based software platform for plant modelling. *Functional Plant Biology* **35**: 751.

Prusinkiewicz P. 2004. Modeling plant growth and development. *Current Opinion in Plant Biology* **7**: 79–83.

Rapilly F. 1991. *L'épidémiologie en pathologie végétale: mycoses aériennes*. Paris: Editions Quae (INRA).

Rapilly F, Jolivet E. 1976. Construction d'un modèle (episept) permettant la simulation d'une épidémie de Septoria nodorum BERK. *sur blé*. Revue de statistique appliquée **24**: 31–60.

Robert C, Bancal M-O, Lannou C. 2004. Wheat leaf rust uredospore production on adult plants: influence of leaf nitrogen content and *Septoria tritici* blotch. *Phytopathology* **94**: 712–721.

Robert C, Bancal M-O, Lannou C, Ney B. 2006. Quantification of the effects of *Septoria tritici* blotch on wheat leaf gas exchange with respect to lesion age, leaf number, and leaf nitrogen status. *Journal of Experimental Botany* **57**: 225–234.

Robert C, Fournier C, Andrieu B, Ney B. 2008. Coupling a 3-D virtual wheat (*Triticum aestivum*) plant model with a *Septoria tritici* epidemic model (Septo3-D): a new approach to investigate plant–pathogen interactions linked to canopy architecture. *Functional Plant Biology* **35**: 997–1013.

Room P, Hanan J, Prusinkiewicz P. 1996. Virtual plants: new perspectives for ecologists, pathologists and agricultural scientists. *Trends in Plant Science* **1**: 33–38.

Saudreau M, Sinoquet H, Santin O, *et al*. 2007. A 3-D model for simulating the spatial and temporal distribution of temperature within ellipsoidal fruit. *Agricultural and Forest Meteorology* **147**: 1–15.

Saudreau M, Pincebourde S, Dassot M, Adam B, Loxdale HD, Biron DG. 2013. On the canopy structure manipulation to buffer climate change effects on insect herbivore development. *Trees* **27**: 239–248.

Sinoquet H, Le Roux X, 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 and Environment* **24**: 395–406.

Tuzet A, Wilson JD. 2002. Wind and turbulence in a sparse but regular plant canopy. *Journal of Applied Meteorology* **41**: 573–587.

Tuzet A, Perrier A, Leuning R. 2003. A coupled model of stomatal conductance, photosynthesis and transpiration. *Plant, Cell & Environment* **26**: 1097–1116.

Varlet-Grancher C, Bonhomme R, Sinoquet H. 1993. *Crop structure and light microclimate: characterization and applications*. Paris: INRA.

Vos J, Evers JB, Buck-Sorlin GH, Andrieu B, Chelle M, de Visser PHB. 2010. Functional–structural plant modelling: a new versatile tool in crop science. *Journal of Experimental Botany* **61**: 2101–2115.

Walters DR, Bingham IJ. 2007. Influence of nutrition on disease development caused by fungal pathogens: implications for plant disease control. *Annals of Applied Biology* **151**: 307–324.

White J. 1979. The plant as a metapopulation. *Annual Review of Ecology and Systematics* **10**: 109–145.

Willocquet L, Berud F, Raoux L, Clerjeau M. 1998. Effects of wind, relative humidity, leaf movement and colony age on dispersal of conidia of *Uncinula necator*, causal agent of grape powdery mildew. *Plant Pathology* **47**: 234–242.

Wilson PA, Chakraborty S. 1998. The virtual plant: a new tool for the study and management of plant diseases. *Crop Protection* **17**: 231–239.

Young C, Werner P, West J. 2007. *Understanding Sclerotinia infection in oilseed rape to improve risk assessment and disease escape*. London: Home-Grown Cereals Authority.

Chapitre 7

Discussion et Perspectives

7.1 Discussion

A partir du constat de la faible modularité des modèles FSPM, nous avons étudié différentes stratégies facilitant la réutilisation logicielle. Nous avons ainsi identifié trois éléments conceptuels nous permettant de répondre à cette problématique :

— **OpenAlea** : une architecture logicielle fondée sur des composants logiciels réutilisables et des workflows scientifiques (Chapitre 2) ;
— **PlantGL et les MTGs** : des formalismes pour la représentation multi-échelles de la topologie et de la géométrie des plantes (Chapitre 3) ;
— **Workflows scientifiques d'ordre supérieur** : une stratégie permettant d'executer des workflows d'analyse de données et de simulation en utilisant des opérateurs algébriques et le λ-calcul (Chapitre 4).

Ces trois éléments sont associés au sein d'un système inspiré du système *Tableau Noir*, présenté dans l'introduction. Les différents élément de ce nouveau système sont présentés ci-dessous :

1. **Sources de connaissance** - Dans notre cas, les sources de connaissance sont des composants OpenAlea représentant soit i) des solveurs ou des processus calculant des variables d'états sur la structure ; ii) des simulateurs permettant de simuler la croissance et le développement de la plante à partir des variables d'états calculées précédemment ou iii) des opérateurs de type adaptateurs, transformant des variables d'états pour rendre compatibles différentes sources de connaissance, ou de changement d'échelle, permettant de résoudre des contraintes à une échelle donnée à partir de l'information disponible aux autres échelles (up et down scaling).
2. **Tableau noir** - Cette structure centrale est représentée par la structure arborescente multi-échelles quotientée (MTG). Cette structure capture à la fois

la topologie d'une plante ou d'un peuplement, son organisation à différents niveaux d'organisation, ainsi que l'ensemble des variables d'états mesurées ou calculées lors de la simulation par les différentes sources de connaissance. Différentes stratégies sont disponibles pour associer une information géométrique aux différents éléments topologiques (plongement géométrique), grâce en particulier à la bibliothèque *PlantGL*.

3. **Ordonnancement** - L'ordonnancement est une méthode dont le rôle est d'activer et d'exécuter les différentes sources de connaissance dans un ordre donné. A la différence du système à tableau noir, qui utilise un ordonnacement opportuniste, nous avons décidé de permettre au modélisateur d'expliciter l'ordonnancement à l'aide d'un workflow scientifique. L'ordonnanceur devient donc le modèle de calcul associé au workflow. Au Chapitre 4, le modèle de calcul de dataflow a été étendu pour pouvoir supporter l'ordre supérieur et les évenements discrets, ainsi qu'illustré dans le Chapitre 6.

Le système proposé ici utilise un grand nombre d'abstractions proposées par Charles Krueger (Krueger, 1992) :

— **Langages de haut-niveau** - La plateforme OpenAlea a une architecture centrée langage. A la différence d'autres systèmes, construits sous la forme d'une application et proposant des mécanismes partagés d'intégration et des structures de données partagées, comme la plateforme GroIMP, OpenAlea utilise le langage Python comme un langage d'assemblage. L'utilisation de Python, langage dynamique et interprété de haut-niveau, permet d'assembler des composants hétérogènes à la volée, durant l'exécution, tout en garantissant la performance à l'exécution du fait de son mécanisme d'extension avec des langages compilés (Pradal et al., 2008 ; Fernique et Pradal, 2018). L'intégration de sources de connaissance disponibles dans différents langages permet en outre de bénéficier de l'expertise scientifique existante dans ces composants sous une forme exécutable, sans avoir besoin de la traduire en langage informatique à partir des informations souvent incomplètes disponibles dans les publications (Pradal, Varoquaux et Langtangen, 2012). Finalement, le langage L-Py est utilisé comme un langage déclaratif de très haut-niveau (Boudon et al., 2012), pour permettre d'exprimer les règles de croissance d'une plante de façon concise et à partir de concepts proches de la botanique.

— **Composants logiciels** - Les sources de connaissance sont représentées par des composants dans OpenAlea. Ce concept étant absent du langage Python, nous l'avons ajouté à la plateforme. Nous définissons un composant comme un objet fonctionel (i.e. functor) avec des entrées et des sorties explicites, typées et annotées (voir chapitre 4). Un composant peut soit être utilisé comme tout

objet Python ou être assemblé au sein d'un workflow, qui est défini lui-même comme un composant. Les composants sont fournis sous la forme de paquets logiciels. Un gestionnaire de paquets permet de les lister, de les découvrir et de faire des recherches à partir des annotations disponibles (nom, entrées/sorties, documentation).

— **Schémas logiciels** - On peut apparenter les workflows à des schémas logiciels exécutables. C'est en explicitant les relations entre les composants au sein d'un workflow que des algorithmes d'ordonnancement (i.e. modèles de calcul) peuvent être développés et ce indépendamment des composants qui composent le workflow. Cette abstraction permet d'améliorer les stratégies d'exécution, pour distribuer les calculs sur de grandes infrastructures distribuées par exemple (Pradal et al., 2017), sans avoir à modifier le schéma d'assemblage de ces composants.

— **Générateur d'application** - *OpenAlea* fournit une application de programmation visuelle, *VisuAlea*. Cette application permet à la fois d'éditer et d'exécuter les workflows. Mais elle peut aussi être considérée, non comme une application en tant que telle, mais comme un générateur d'application. En effet, l'interface graphique de chaque composant ou activité est indépendante des autres. Elle est générée automatiquement à partir du type des entrées du composant. A chaque type est associé une interface graphique. Ainsi l'interface graphique du composant peut être générée automatiquement par la composition des interfaces de ces entrées. *VisuAlea* permet de construire dynamiquement une application par programmation visuelle, en assemblant dynamiquement des composants pour former un workflow. Et, à partir de ce workflow, une application, un script ou un notebook Jupyter peuvent être générés (Pradal et al., 2015 ; Pradal et al., 2017).

— **Systèmes de réécriture** - Les L-systèmes sont des systèmes de réécriture dont l'abstraction est très présente dans OpenAlea. D'autres formes moins évidentes de réécriture sont utilisées, pour définir les workflows d'ordre supérieur. En effet, la transformation d'un sous-workflow en λ-expression est une forme de réécriture. D'autres formes de réécriture ont été étudiées, en collaboration avec Sarah Cohen-Boulakia, pour pouvoir générer l'information de provenance lors de l'utilisation d'opérateurs algébriques.

Ces opérateurs (e.g. map, reduce) permettent de distribuer de façon transparente pour l'utilisateur les calculs, mais faussent la provenance des données. Par exemple, un opérateur *map* prend en entrée un ensemble de données ainsi qu'une fonction et retourne un ensemble de données, dont chaque sortie a été calculée à partir de chaque entrée. Par réécriture de la provenance, on peut

indiquer que chaque sortie ne dépend que d'une entrée. Autrement, chaque sortie dépendrait de toutes les données d'entrée.

— **Architecture logicielle** - En complément de l'architecture de la plateforme OpenAlea et de PlantGL, s'ajoute une architecture issue de l'architecture des tableaux noirs. Cette architecture nous permet de mettre en relation différents concepts afin de répondre de façon conceptuelle et pratique aux problèmes de réutilisation et de modularité pour simuler les modèles complexes structure-fonction.

Finalement, à ces abstractions informatiques s'ajoute le formalisme mathématique des MTGs qui est central dans ce travail.

Dans cette thèse, ce formalisme a été appliqué à la modélisation et à la simulation des plantes et plus seulement à la représentation, à la mesure et à l'analyse de celles-ci. S'appuyer sur un standard de représentation de l'architecture des plantes à différentes échelles a facilité à la fois la communication entre les différents composants logiciels et le lien entre la mesure et la simulation, tout en simplifiant la validation des modèles et l'utilisation de processus comme l'interception de la lumière sur des plantes reconstruites (Chen et al., 2018).

Le système proposé, qui est un des fondements de la plateforme OpenAlea, a été illustré par deux cas contrastés :

— Dans le chapitre 5, nous avons extrait à partir de modèles de simulation complexes une source de connaissance spécifique, pouvant être étendue et réutilisée dans diverses applications. A partir d'un modèle de feuille, publié par Prévot *et al.* (Prévot, Aries et Monestiez, 1991), et implémenté dans Adel (Fournier et Andrieu, 1999), nous avons proposé un nouveau modèle géométrique de feuille de graminées en 3D se développant au cours du temps et dont la forme est déformée par opérateurs morphologiques modélisant l'impact de stress physiologique sur la forme en croissance. Ce modèle est actuellement utilisé dans de nombreux modèles de graminés au sein de la plateforme OpenAlea (Fournier et al., 2010).

— Dans le chapitre 6, nous avons conçu un système permettant de modéliser de façon générique le développement et la dispersion de maladies foliaires au sein d'un peuplement de plantes simulé par des modèles structure-fonction en interaction avec des modèles micro-climatiques. Ce système a permis de simuler le patho-système blé / septoriose et vigne /oïdium. Il a par la suite été étendu pour pouvoir représenter des complexes de maladies foliaires entrant en compétition (complexe rouille/septoriose du blé) (Garin et al., 2018) et a été validé à partir de données expérimentales (Robert et al., 2018).

Aujourd'hui, ce schéma conceptuel est utilisé dans de nombreux projets en France (Fournier et al., 2010 ; Boudon et al., 2012 ; Garin et al., 2014 ; Garin et al., 2018 ; Reyes et al., 2018 ; Chen et al., 2018 ; Barillot et al., 2018 ; Perez et al., 2019 ; Albasha et al., 2019). Il commence aussi a être utilisé à l'étranger à travers des collaborations (Long et al., 2018), des projets européens (e.g. le projet européen H2020 IPM DSS) ou des projets internationaux comme AMEI (Martre et al., 2018) ou CropsInSilico (Marshall-Colon et al., 2017b).

Cependant, malgré l'intérêt de notre approche, des limites persistent :

— **Intégration de composants** - La plateforme OpenAlea est centrée autour du langage Python. Or, pour des raisons de performances, il est souvent nécessaire d'intégrer des bibliothèques hétérogènes en C++, ce qui requiert une expertise et un investissement important. Pour dépasser cette limite et faciliter l'intégration de composants C++, nous avons récemment développé AutoWIG, un générateur automatique d'interface Python pour des bibliothèques C++ (Fernique et Pradal, 2018). L'avantage de cette approche, qui repose sur l'infrastructure multi-plateformes de compilation *llvm*, est de générer à partir du graphe de compilation l'ensemble des wrappers permettant d'accéder à l'ensemble des classes et des méthodes depuis Python. Cependant, l'utilisation de cet outil est encore limité aux informaticiens, plutôt qu'aux biologistes modélisateurs.

— **Communication par mémoire partagée** - D'autres systèmes sont en train d'être développés, comme CropsInSilico. Dans ce cas, les composants communiquent non pas par mémoire partagée au sein du même ordinateur, mais de façon distribuée à partir de messages (Marshall-Colon et al., 2017b), favorisant ainsi un couplage faible entre composants. L'avantage de cette approche est de simplifier le travail d'intégration et de généraliser la notion de composant. De plus, cette approche est nécessaire pour passer à l'échelle et utiliser des infrastructures distribuées. Son inconvénient est que les composants ne partagent pas de structure de données, et que la communication est plus lente lors de l'échange de structures complexes en mémoire.

— **Simulation centralisée de systèmes complexes** - Nous avons vu comment l'utilisation d'opérateurs algébriques permettait de distribuer les calculs de façon transparente en fonction des données (Pradal et al., 2015 ; Pradal et al., 2017 ; Pradal et al., 2018). Cependant, la distribution n'est efficace que si l'on effectue le même traitement sur un très grand jeu de données comme en analyse de sensibilité ou en phénotypage haut-débit, où l'on reconstruit un grand nombres d'architecture de plantes à partir d'images. Lors de simulations complexes, du fait de la dépendance locale de l'état de la simulation au pas

de temps précédent et des rétro-actions, en l'état actuel des connaissances, les workflows ne permettent pas de distribuer les calculs de façon efficace.

— **Absence de sémantique** - L'utilisation d'une structure centrale permet l'intégration de sources de connaissance et de la modularité. Ainsi, chaque source de connaissance doit être adaptée à cette structure de données centrale et donc en dépend. De plus, le partage d'information entre les sources de connaissances se fait de façon empirique et syntaxique, sans recourir à des notions de sémantique. Si une source de connaissance calcule l'interception de la lumière, elle doit ajouter une propriété nommée 'absorbed energy' qui sera accedée par une source de connaissances photosynthèse. Si le nom de cette propriété change, il faudra ajouter des adaptateurs entre ces sources de connaissance pour qu'elles puissent échanger cette information. Il manque donc un niveau d'abstraction pour faire du mapping sémantique entre sources de connaissances et proposer des standards d'échanges d'information et d'annotation des modèles.

— **Compatibilité faible entre L-systems et MTG** - La structure centrale multi-échelles permet d'accéder aux données et de calculer de façon efficace à différentes échelles de représentation. Des algorithmes génériques ont été développés pour pouvoir la parcourir, afin de résoudre des contraintes géométriques ou de calculer de façon efficace l'allocation de carbone par exemple (Reyes et al., 2018). Récemment, le formalisme des L-systems a été étendu pour pouvoir réécrire un MTG en transformant cette structure en *l-string* (Boudon et al., 2012), afin d'exprimer le développement d'une plante à l'aide d'un langage déclaratif Lsystem, plus simple qu'un graphe multi-échelles. Mais cela impose la copie et la transformation de la structure à chaque pas de temps. Une évolution des MTG serait de fournir un système de réécriture directement sur MTG, un peu à la façon de GroIMP, pour augmenter la compatibilité entre les formalismes et faire apparaître des modèles à plus forte généricité.

— **Représentation en mémoire du MTG** - Une autre limite est que le MTG est partagé en mémoire, ce qui garantit des performances intéressantes pour la simulation. Mais cela limite les potentialités de passage à l'échelle dans un contexte où la mémoire est limitée (out of memory). Avec l'avènement des bases de données non conventionnelles de type graphe (comme Neo4J, DEX ou G-Sparks) (Angles et Gutierrez, 2008), il serait intéressant de tester la possibilité de représenter un MTG sous la forme d'une base de données. Cela favoriserait l'indépendance des sources de connaissance avec l'implémentation du MTG. La communication, se faisant sous la forme de requêtes, faciliterait alors la constitution de bases de données publiques sur différentes

architectures, le calcul distribué et la visualisation à grande échelle.

— **Environnement de modélisation mono-paradigme** - Il existe différents paradigmes de programmation pour modéliser une plante : la programmation procédurale et impérative, la programmation déclarative en utilisant les langages de réécritures, la programmation visuelle en constituant des workflows scientifiques. Tous ces paradigmes ne sont pas équivalents. Ils ont chacun leurs avantages en fonction de la tâche à effectuer. La plupart des processus, ainsi que la plateforme elle-même, sont implémentés en langage objet. Les modèles de croissance de l'architecture sont souvent en Lsystems, alors que l'assemblage de composants se fait majoritairement en programmation visuelle dans OpenAlea. Or, chaque environnement de modélisation favorise un paradigme au détriment des autres, même s'ils coexistent tous ensemble sous la forme de bibliothèques Python. Il serait pertinent de proposer un environnement permettant de modéliser les plantes en utilisant de façon conjointe différents paradigmes de modélisation.

7.2 Perspectives

A partir des résultats obtenus dans cette thèse, de nouvelles directions de recherche restent à explorer. Parmi les nombreux défis scientifiques à relever, trois perspectives principales se dessinent qui renouvellent la thématique de cette thèse : i) la modélisation structure-fonction en peuplement mixte, intégrant parties aérienne et racinaire, ii) la standardisation des modèles et leur annotation sémantique et iii) le phénotypage haut-débit dirigé par les modèles.

7.2.1 Modélisation structure-fonction de peuplements mixtes

Les modèles structure-fonction se démarquent d'autres formalismes en prenant en compte la spatialisation des processus et en explicitant la structure de la plante. Ces modèles ont surtout été utilisés pour comprendre les phénomènes complexes de fonctionnement et de développement de la plante, plutôt que pour prédire sa production. Or, dans un contexte de changements globaux et de développement durable, une demande apparaît pour étudier des systèmes plus complexes comme les peuplements mixtes, qui permettent d'utiliser les complémentarités entre plantes pour améliorer la production en limitant le recours à des intrants et à des pesticides. La modélisation de mélange variétaux nécessite de prendre en compte l'hétérogénéité des plantes en mélange et leur structure pour simuler la compétition pour l'acquisition de ressources au niveau aérien (e.g. lumière) et racinaire (e.g. nutriments) (Gaudio et al.,

2019). Les modèles de culture ne sont aujourd'hui pas capables de représenter de tels systèmes hétérogènes spatialement, car ils supposent que le peuplement de plantes, dans une parcelle, est constitué de plantes similaires, calculées à partir d'une plante moyenne. Une des perspectives actuelle pour les FSPMs est de modéliser des peuplements mixtes, racinaires et aériens, en relation avec le sol et les micro-organismes pour favoriser la compréhension de l'influence du mélange mixte dans la croissance (compétition pour la lumière et qualité de la lumière), ainsi que dans la propagation des maladies (Evers et al., 2018; Garin et al., 2018; Gaudio et al., 2019). Mais pour relever ce défi, plusieurs verrous sont à lever :

— **Quel schéma conceptuel utiliser pour modéliser les interactions et le fonctionnement des parties aérienne et racinaire, en interaction avec l'environnement ?** Actuellement, la plupart des modèles ne prennent en compte qu'une partie du système et de son fonctionnement en structure-fonction. Mais en plus de devoir coupler appareils racinaire et aérien, il s'agit aussi de représenter au sein d'un même modèle différentes variétés et différentes plantes de façon générique et réutilisable.

— **Comment préserver la modularité d'un système FSPM plante / sol / atmosphère ?** A la représentation de la plante dans son ensemble, s'ajoute la modélisation de son interaction avec son environnement. Le schéma conceptuel que nous proposons dans cette thèse peut facilement être adapté aux contraintes biotiques et abiotiques en peuplement mixte au niveau aérien. Cependant, il n'existe pas de modèle modulaire permettant de coupler un modèle pour les parties aériennes, un autre pour les parties racinaires avec un modèle de flux d'eaux et de nutriments dans un sol spatialisé, modélisant l'environnement des parties aériennes.

— **Quel formalisme de simulation utiliser pour simuler des centaines de plantes en interactions ?** - Cette question est posée dans les FSPMs modélisant des populations de plantes (Garin et al., 2014; Garin et al., 2018; Barillot et al., 2018). Dès que l'on représente des centaines de plantes en interaction, se pose le problème du temps de simulation. Même pour des plantes annuelles, dont la structure est simple, le nombre d'entités composant le système est élevé. Pour représenter un mètre carré de blé, il faut simuler entre 100 et 200 plantes, comportant entre 10 et 20 feuilles. Si l'on simule des processus explicites, comme les maladies foliaires, où l'on représente les lésions foliaires par un agent (Garin et al., 2014), le nombre de composants du système peu rapidement se chiffrer en centaines de milliers d'entités en interactions, en considérant 100 lésions par feuille. La complexité augmente si l'on considère

non plus des entités homogènes, fonctionnant de façon similaire, mais hétérogènes, comme différentes plantes et différentes maladies. Il faut pouvoir modéliser chacune des entités, son fonctionnement mais aussi ses interactions. Or, à la différence de l'analyse de grands volumes de données, facilement parallélisables, peu de travaux ont été consacrés à la parallélisationde systèmes contenant des entités spatialisées et en interaction, au moins dans le domaine de la modélisation et de la simulation des plantes. En effet, il faut modéliser à la fois la dépendance spatiale (les plantes voisines en interaction avec la plante étudiée) et la dépendance temporelle (le fonctionnement et la structure de la plante liés à son état précédent). La question du formalisme à utiliser se pose donc pour considérer un très grand nombre de plantes. Faut-il utiliser des approches de méta-modélisation où un modèle explicite (mécaniste) peut être résumé par un modèle statistique et utilisé à une échelle plus grande (Garin et al., 2018) ? Faut-il avoir une approche multi-échelles pour considérer les interactions entre les plantes en fonction de leur voisinage proche, découper le problème en zones indépendantes mais recouvrantes et distribuer les calculs ?

7.2.2 Standardisation et annotation sémantique des modèles

Un des apports de la plateforme OpenAlea a été de faciliter le découpage de modèles en composants pour pouvoir les assembler dans un système de workflow scientifique afin de les recomposer pour former des applications dédiées. Ce découpage a permis aux modélisateurs de définir explicitement les entrées-sorties de leurs modèles, le type associé ainsi que de nombreuses méta-informations (auteur, license, documentation, …) (Pradal et al., 2008). Cependant, bien que ce travail d'annotation facilite par exemple la découverte et la recherche de nouveaux composants à partir de la documentation ou des données d'entrées, ainsi que la génération automatique d'interface graphique associée, cela n'est pas suffisant pour permettre une interopérabilité entre plateformes de modélisation. En effet, les données d'entrées ne sont pas normalisées et les modèles sont exprimés soit dans un langage haut-niveau ou dans un langage compilé. Or, dans tous les cas, comprendre un modèle à travers son implémentation est difficile, même pour ceux qui connaissent la syntaxe du langage source. Cela est du au fait que l'implémentation contient un ensemble d'instructions nécessaires à l'exécution, mais indépendante du formalisme de modélisation utilisé dans une publication, par exemple. Deux directions s'offrent à nous : d'une part i) la standardisation des structures de données utilisées par les modèles et d'autre part ii) la standardisation des modèles eux-mêmes à travers un formalisme proche du formalisme mathématique, qui est la meilleure abstraction disponible (Pradal, Varoquaux

et Langtangen, 2012).

A propos de la standardisation des structures de données, un premier effort a été fait lors de la publication du formalisme des MTGs (Godin et Caraglio, 1998 ; Godin, 2000). Cependant, peu de plateformes de modélisation ont implémenté ce formalisme du fait de sa complexité. Récemment cependant, une forme simplifiée a été élaborée, à travers le format RSML (Root System Markup Langage) (Lobet et al., 2015), pour permettre de représenter la topologie et la géométrie des systèmes racinaires. Ce format a été implémenté dans différentes plateformes de phénotypage racinaire, différents modèles de simulation racinaire et de modèles de sol, ainsi que des outils d'analyse. Une implémentation en C++, C#, Python et R a été publiée. Plusieurs ontologies (crop et plant ontology) ont été utilisées pour décrire les types racinaires et leurs propriétés associées. Il reste cependant à généraliser cette initiative à la plante entière et à mobiliser la communauté internationale pour faire émerger un standard.

Pour ce qui est de l'annotation des modèles, plusieurs intiatives en biologie des systèmes ont développé des langages déclaratifs (e.g. SBML, CellML, NeuroML) pour décrire les modèles à partir d'équations mathématiques et stoechiométriques. Une revue récente, dirigée par le Pr. Millar décrit l'ensemble de ces initiatives en biologie (Millar et al., 2019). En modélisation des cultures, une initiative similaire a été intiée, l'intiative internationale Agriculture Modelling Exchange Initiative ou AMEI (Martre et al., 2018), regroupant les principaux acteurs de ce domaine (ApSim, DSSAT, BioMa, SimPlace, RECORD, OpenAlea). L'objet de cette initiative est de définir un langage commun de modélisation pour pouvoir partager des modèles entre différentes plateformes implémentées dans différents langages de programmation. Pour cela, le langage Crop2ML a été défini. Il décrit la structure d'un modèle atomique et composite sous la forme d'un workflow, ainsi que le coeur du modèle en utilisant un langage partagé, un langage métier (Domain Specific Language ou DSL) proche du langage Python et typé. A partir de ce DSL, les différentes implémentations dans les différentes plateformes sont générées automatiquement par transcodage. Cette stratégie est un pari en cours (thèse de Cyrille Midingoyi) et nécessitera d'être étendue aux FSPMs.

7.2.3 Phénotypage haut-débit dirigé par les modèles

Finalement, la dernière perspective que nous voulons évoquer concerne le phénotypage haut-débit des plantes. Cet autre domaine, distinct de la modélisation est en plein expansion. Bien sur, le phénotypage, c'est à dire l'acquisition de la géométrie et/ou de la topologie des plantes, existe depuis des décennies. Mais le développement de capteurs et des plateformes automatisées permet d'obtenir un débit de données

sans précédent et en constante augmentation. Plusieurs questions se posent dans ce domaine. Un des enjeux est d'automatiser intégralement l'acquisition et le traitement de cette masse de données complexe, structurée et volumineuse. Avec le développement des méthodes d'apprentissage, cette limite sera bientôt dépassée. Cependant, de nombreux traits fonctionnels ne sont pas directement mesurables comme l'efficience de l'interception de la lumière (LUE). Le recours à des modèles bio-physiques permet de calculer des traits à partir de l'architecture reconstruite (Cabrera-Bosquet et al., 2016 ; Chen et al., 2018 ; Perez et al., 2019). De façon orthogonale, l'utilisation de méthodes de simulation permet de simuler de grands jeux de données et ainsi d'entraîner des méthodes d'apprentissage comme des réseaux profonds par exemple (Ubbens et al., 2018 ; Reichstein et al., 2019). Une possiblité serait d'utiliser ces méthodes pour segmenter les différents organes de plante, en utilisant des modèles de simulation de l'architecture afin de générer de la donnée synthétique, et donc diminuer la base d'apprentissage, ainsi que pour réaliser des réseaux adversaires (adverserial networks) et auto-apprenants. Finalement, la dernière question ouverte est de savoir si ces méthodes ne vont pas obliger à repenser le rôle des modèles actuels face à des techniques pouvant assimiler un grand volume de données et ayant de bien meilleur capacités prédictives.

Bibliographie

Ackerman, WB (1982). « Data Flow Languages ». In : *Computer* 15.2, p. 15-25.

Albasha, Rami et al. (2019). « HydroShoot : a functional-structural plant model for simulating hydraulic structure, gas and energy exchange dynamics of complex plant canopies under water deficit-application to grapevine (Vitis vinifera L.) » In : *bioRxiv*, p. 542803.

Alexandrescu, Andrei (2001). *Modern C++ design : generic programming and design patterns applied*. Addison-Wesley.

Allen, M. T., P. Prusinkiewicz et T. M. DeJong (2005). « Using L-systems for modeling source-sink interactions, architecture and physiology of growing trees : the L-PEACH model ». In : *New Phytologist* 166.3, p. 869-880. issn : 1469-8137. doi : 10.1111/j.1469-8137.2005.01348.x. url : http://dx.doi.org/10.1111/j.1469-8137.2005.01348.x.

Angles, Renzo et Claudio Gutierrez (2008). « Survey of graph database models ». In : *ACM Computing Surveys (CSUR)* 40.1, p. 1.

Assayag, Gérard et al. (1999). « Computer-assisted composition at IRCAM : From PatchWork to OpenMusic ». In : *Computer Music Journal* 23.3, p. 59-72.

Backus, John (1978). « Can Programming Be Liberated from the von Neumann Style ? A Functional Style and Its Algebra of Programs ». In : *Communications*.

Balduzzi, Mathilde et al. (2017). « Reshaping Plant Biology : Qualitative and Quantitative Descriptors for Plant Morphology. » In : *Front. Plant Sci.* 8, p. 117. doi : 10.3389/fpls.2017.00117. url : http://www.ncbi.nlm.nih.gov/pubmed/28217137http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5289971.

Barillot, Romain et al. (2018). « Investigation of complex canopies with a functional–structural plant model as exemplified by leaf inclination effect on the functioning of pure and mixed stands of wheat during grain filling ». In : *Annals of Botany*, mcy208. doi : 10.1093/aob/mcy208. eprint : /oup/backfile/content_public/journal/aob/pap/10.1093_aob_mcy208/3/mcy208.pdf. url : http://dx.doi.org/10.1093/aob/mcy208.

Baroth, Ed et Chris Hartsough (1995). « Visual programming in the real world ». In : *Visual object-oriented programming*. Manning Publications Co., p. 21-42.

Barthélémy, Daniel et Yves Caraglio (2007). « Plant architecture : A dynamic, multilevel and comprehensive approach to plant form, structure and ontogeny ». In : *Annals of Botany* 99.3, p. 375-407.

Behnel, Stefan et al. (2011). « Cython : The best of both worlds ». In : *Computing in Science & Engineering* 13.2, p. 31-39.

Bic, Lubomir (1990). « A process-oriented model for efficient execution of dataflow programs ». In : *Journal of Parallel and Distributed Computing* 8.1, p. 42-51.

Boudon, F. et al. (2006). « Estimating the fractal dimension of plants using the two-surface method. An analysis based on 3D-digitized tree foliage ». In : *Fractals* 14.3, p. 149-163.

Boudon, Frédéric et al. (2012). « L-py : an L-system simulation framework for modeling plant architecture development based on a dynamic language. » In : *Frontiers in plant science* 3. issn : 1664-462X. doi : 10.3389/fpls.2012.00076. url : http://dx.doi.org/10.3389/fpls.2012.00076.

Bucksch, Alexander et al. (2017). « Morphological plant modeling : Unleashing geometric and topological potential within the plant sciences ». In : *Frontiers in Plant Science* 8, p. 900. issn : 1664-462X. doi : 10.3389/FPLS.2017.00900. url : http://journal.frontiersin.org/article/10.3389/fpls.2017.00900/full.

Buxton, John N et Brian Randell (1970). *Software Engineering Techniques : Report on a Conference Sponsored by the NATO Science Committee*. NATO Science Committee ; available from Scientific Affairs Division, NATO.

Cabrera-Bosquet, Llorenç et al. (2016). « High-throughput estimation of incident light, light interception and radiation-use efficiency of thousands of plants in a phenotyping platform ». In : *New Phytologist* 212.1, p. 269-281.

Callahan, Steven P et al. (2006). « VisTrails : visualization meets data management ». In : *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. ACM, p. 745-747.

Cerezo, Nadia, Johan Montagnat et Mireille Blay-Fornarino (2013). « Computer-assisted scientific workflow design ». In : *Journal of grid computing* 11.3, p. 585-612.

Chelle, Michaël et Bruno Andrieu (1999). « Radiative models for architectural modeling ». In : *Agronomie* 19.3-4, p. 225-240.

Chen, Tsu-Wei et al. (2018). « Genetic and environmental dissection of biomass accumulation in multi-genotype maize canopies ». In : *Journal of experimental botany*.

Chitwood, Daniel H et al. (2012). « Leaf asymmetry as a developmental constraint imposed by auxin-dependent phyllotactic patterning ». In : *The Plant Cell* 24.6, p. 2318-2327.

Cieslak, Mikolaj, Alla N. Seleznyova et Jim Hanan (avr. 2011). « A functional-structural kiwifruit vine model integrating architecture, carbon dynamics and effects of the environment ». In : *Annals of Botany* 107.5, p. 747-764. doi : 10.1093/aob/mcq180. url : http://dx.doi.org/10.1093/aob/mcq180.

Cohen-Boulakia, Sarah (2015). « Data Integration in the Life Sciences : Scientific Workflows, Provenance, and Ranking ». Thèse de doct. Université Paris-Sud.

Cohen-Boulakia, Sarah et al. (2017). « Scientific workflows for computational reproducibility in the life sciences : Status, challenges and opportunities ». In : *Future Generation Computer Systems* 75, p. 284-298.

Corkill, Daniel D (2003). « Collaborating Software : Blackboard and Multi-Agent Systems & the Future ». In : *Proceedings of the International Lisp Conference*. New York, New York. url : http://mas.cs.umass.edu/paper/265.

Costes, E. et al. (2008). « MAppleT : simulation of apple tree development using mixed stochastic and biomechanical models ». In : *Functional Plant Biology* 35.9 & 10, p. 936-950. url : http://www-sop.inria.fr/virtualplants/Publications/2008/CSRGPG08.

Cuevas-Vicenttín, Víctor et al. (2012). « Scientific Workflows and Provenance : Introduction and Research Opportunities ». In : *Datenbank-Spektrum* 12.3, p. 193-203. issn : 1618-2162, 1610-1995. doi : 10.1007/s13222-012-0100-z. arXiv : arXiv:1311.4610v2. url : http://link.springer.com/10.1007/s13222-012-0100-z.

Da Silva, D. et al. (2006). « A Critical Appraisal of the Box Counting Method to Assess the Fractal Dimension of Tree Crowns. » In : *Lecture Notes in Computer Sciences (Proceedings of the 2nd International Symposium on Visual Computing)* 4291, p. 751-750.

Davidson, Susan B et al. (2007). « Provenance in scientific workflow systems. » In : *IEEE Data Eng. Bull.* 30.4, p. 44-50.

Davis, Alan L (1978). « The architecture and system method of DDM1 : A recursively structured data driven machine ». In : *Proceedings of the 5th annual symposium on Computer architecture*. ACM, p. 210-215.

Deelman, Ewa et al. (2005). « Pegasus : A framework for mapping complex scientific workflows onto distributed systems ». In : *Scientific Programming* 13.3, p. 219-237.

Deelman, Ewa et al. (2009). « Workflows and e-Science : An overview of workflow system features and capabilities ». In : *Future generation computer systems* 25.5, p. 528-540.

Dennis, Jack B (1980). « Data flow supercomputers ». In : *Computer* 11, p. 48-56.

Di Tommaso, Paolo et al. (2017). « Nextflow enables reproducible computational workflows ». In : *Nature biotechnology* 35.4, p. 316-319.

Doussan, Claude, Loïc Pagès et Gilles Vercambre (1998). « Modelling of the hydraulic architecture of root systems : an integrated approach to water absorption—model description ». In : *Annals of botany* 81.2, p. 213-223.

Engelmore, Robert et Tony Morgan, éd. (1988). *Blackboard Systems (The Insight Series in Artificial Intelligence)*. First Edition. Addison-Wesley Pub (Sd). isbn : 0201174316. url : http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0201174316.

Evers, Jochem B et al. (2018). « Understanding and optimizing species mixtures using functional–structural plant modelling ». In : *Journal of experimental botany*.

Fernique, Pierre et Christophe Pradal (2018). « AutoWIG : automatic generation of python bindings for C++ libraries ». In : *PeerJ Computer Science* 4, e149.

Ferraro, P., C. Godin et P. Prusinkiewicz (2005). « Toward a quantification of self-similarity in plants ». In : *Fractals* 13.2, p. 91-109.

Fourcaud, Thierry et al. (mai 2008). « Plant Growth Modelling and Applications : The Increasing Importance of Plant Architecture in Growth Models ». In : *Annals of Botany* 101.8, p. 1053-1063. issn : 1095-8290. doi : 10.1093/aob/mcn050. url : http://dx.doi.org/10.1093/aob/mcn050.

Fournier, C. et B. Andrieu (1998). « A 3D Architectural and Process-based Model of Maize Development ». In : *Annals of Botany* 81.2, p. 233-250. doi : 10.1006/anbo.1997.0549. eprint : http://aob.oxfordjournals.org/content/81/2/233.full.pdf+html. url : http://aob.oxfordjournals.org/content/81/2/233.abstract.

Fournier, C. et al. (2003). « ADEL-wheat : a 3D architectural model of wheat development ». In : *Plant Growth Modeling and Applications*. Sous la dir. de Bao-Gang Hu et Marc Jaeger. Proceedings - PMA03, 2003' International Symposium on plant Growth Modeling, Simulation, visualization and their Applications. Beijing, China, october 13-16, 2003. Beijing, China : Tsinghua University Press et Springer, p. 54-66.

Fournier, C. et al. (2010). « Building modular FSPM under OpenAlea : concepts and applications ». In : *6th International Workshop on Functional-Structural Plant Models*. Sous la dir. de D. Da Silva T. de Jong, p. 109-112. url : http://www-sop.inria.fr/virtualplants/Publications/2010/FPLCSLBC10.

Fournier, Christian et Bruno Andrieu (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 ». In : *Agronomie*

19.3-4, p. 313-327. issn : 0249-5627. doi : 10.1051/agro:19990311. url : http://dx.doi.org/10.1051/agro:19990311.

Fournier, Christian et al. (2016). « Toward the inter-comparison of radiation transfert model for plant modelling application ». In : *IEEE International Conference on Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications (FSPMA 2016)*.

Freeman, Peter (1983). « Reusable software engineering : Concepts and research directions ». In : *ITT Proceedings of the Workshop on Reusability in Programming*. T. 129, p. 137.

Fumey, Damien et al. (2011). « How young trees cope with removal of whole or parts of shoots : an analysis of local and distant responses to pruning in 1-year-old apple (Malus× domestica ; Rosaceae) trees ». In : *American journal of botany* 98.11, p. 1737-1751.

Gamma, E. et al. (1995). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Garin, Guillaume et al. (2014). « A modelling framework to simulate foliar fungal epidemics using functional–structural plant models ». In : *Annals of Botany* 114.4, p. 795-812. doi : 10.1093/aob/mcu101. eprint : /oup/backfile/content_public/journal/aob/114/4/10.1093_aob_mcu101/2/mcu101.pdf. url : http://dx.doi.org/10.1093/aob/mcu101.

Garin, Guillaume et al. (2018). « Modelling interaction dynamics between two foliar pathogens in wheat : a multi-scale approach ». In : *Annals of botany* 121.5, p. 927-940.

Gaudio, Noémie et al. (2019). « Current knowledge and future research opportunities for modeling annual crop mixtures. A review ». In : *Agronomy for Sustainable Development* 39.2, p. 20.

Goble, Carole A et al. (2010). « myExperiment : a repository and social network for the sharing of bioinformatics workflows ». In : *Nucleic acids research* 38.suppl_2, W677-W682.

Godin, C. (2000). « Representing and encoding plant architecture : a review ». In : *Annals of Forest Science* 57.05-juin, p. 413-438.

Godin, Christophe et Yves Caraglio (1998). « A multiscale model of plant topological structures ». In : *Journal of Theoretical Biology* 191, p. 1-46.

Godin, Christophe, Evelyne Costes et Hervé Sinoquet (2005). « Plant architecture modelling - virtual plants and complex systems ». In : *Plant Architecture and its Manipulation*. Sous la dir. de C. Turnbull. T. 17. Annual plant reviews. Blackwell, p. 238-287.

Godin, Christophe et Yann Guédon (1999). « AMAPmod introduction and reference manual ». In : *CIRAD : Marie-Hélène Lafond*.

Goecks, Jeremy, Anton Nekrutenko et James Taylor (2010). « Galaxy : a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences ». In : *Genome biology* 11.8, R86.

Görlach, Katharina et al. (2011). « Conventional workflow technology for scientific simulation ». In : *Guide to e-Science*. Springer, p. 323-352.

Guédon, Y., P. Heuret et E. Costes (déc. 2003). « Comparison methods for branching and axillary flowering sequences ». In : *Journal of Theoretical Biology* 225.3, p. 301-325. issn : 00225193. doi : 10.1016/S0022-5193(03)00249-2. url : http://dx.doi.org/10.1016/S0022-5193(03)00249-2.

Guédon, Y. et al. (oct. 2001). « Pattern Analysis in Branching and Axillary Flowering Sequences ». In : *Journal of Theoretical Biology* 212.4, p. 481-520. issn : 00225193. doi : 10.1006/jtbi.2001.2392. url : http://dx.doi.org/10.1006/jtbi.2001.2392.

Guédon, Yann, Daniel Barthélémy et Yves Caraglio (1999). « Analyzing spatial structures in forests tree architectures ». In : *Empirical and process-based models for forest tree and stand growth simulation*. Sous la dir. d'A. Amaro et M. Tmoé. IUFRO. Oeiras (Portugal), p. 23-42.

Guédon, Yann et Evelyne Costes (1998). « A statistical appraoch for analyzing sequences in fruit tree architecture ». In : *Fifth international symposium on computer modelling in fruit research and orchard management*.

Guédon, Yann et al. (oct. 2007). « Analyzing growth components in trees ». In : *Journal of Theoretical Biology* 248.3, p. 418-447. issn : 00225193. doi : 10.1016/j.jtbi.2007.05.029. url : http://dx.doi.org/10.1016/j.jtbi.2007.05.029.

Halley, J. M. et al. (2004). « Uses and abuses of fractal methodology in ecology ». In : *Ecology Letters* 7.3, p. 254-271.

Hanan, Jim et Przemyslaw Prusinkiewicz (2008). « Foreword : studying plants with functional–structural models ». In : *Functional Plant Biology* 35.10, p. vi-viii.

Hemmerling, Reinhard et al. (2008). « The rule-based language XL and the modelling environment GroIMP illustrated with simulated tree competition ». In : *Functional Plant Biology* 35.10, p. 739+. issn : 1445-4408. doi : 10.1071/FP08052. url : http://dx.doi.org/10.1071/FP08052.

Hemmerling, Reinhard et al. (mar. 2013). « Extension of the GroIMP modelling platform to allow easy specification of differential equations describing biological processes within plant models ». In : *Computers and Electronics in Agriculture* 92,

p. 1-8. issn : 01681699. doi : 10.1016/j.compag.2012.12.007. url : http://dx.doi.org/10.1016/j.compag.2012.12.007.

Henke, Michael, Winfried Kurth et Gerhard H Buck-Sorlin (2016). « FSPM-P : towards a general functional-structural plant model for robust and comprehensive model development ». In : *Frontiers of Computer Science* 10.6, p. 1103-1117.

Hess, Roland (2007). *The essential Blender : guide to 3D creation with the open source suite Blender*. No Starch Press.

Hey, Tony, Stewart Tansley, Kristin M Tolle et al. (2009). *The fourth paradigm : data-intensive scientific discovery*. T. 1. Microsoft research Redmond, WA.

Jackson, John E (1980). « Light interception and utilization by orchard systems ». In : *Horticultural Reviews, Volume 2*, p. 208-267.

Jacob, François (1987). *La logique du vivant : une histoire de l'hérédité*. http ://www.scientificcommons.org/ Gallimard. url : http://hdl.handle.net/2042/29885.

Jacob, Joseph C et al. (2009). « Montage : a grid portal and software toolkit for science-grade astronomical image mosaicking ». In : *International Journal of Computational Science and Engineering* 4.2, p. 73-87.

Jagannathan, V., Rajendra Dodhiawala et Lawrence S. Baum (1989). *Blackboard Architectures and Applications (Perspectives in Artificial Intelligence)*. Academic Press. isbn : 0123799406. url : http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0123799406.

Jamal, Rahman et Lothar Wenzel (1995). « The applicability of the visual programming language LabVIEW to large real-world applications ». In : *Visual Languages, Proceedings., 11th IEEE International Symposium on*. IEEE, p. 99-106.

Johnston, Wesley M, JR Hanna et Richard J Millar (2004). « Advances in dataflow programming languages ». In : *ACM computing surveys (CSUR)* 36.1, p. 1-34.

Kaplan, Donald R (2001). « The science of plant morphology : definition, history, and role in modern biology ». In : *American Journal of Botany* 88.10, p. 1711-1741.

Kendall, David G (1984). « Shape manifolds, procrustean metrics, and complex projective spaces ». In : *Bulletin of the London Mathematical Society* 16.2, p. 81-121.

Kniemeyer, O., G.-H. Buck-Sorlin et Kurth W. (2004). « A Graph Grammar Approach to Artificial Life ». In : *Artificial Life* 10.4, p. 413-431.

Kniemeyer, Ole et Winfried Kurth (2008). « Applications of Graph Transformations with Industrial Relevance ». In : sous la dir. d'Andy Schürr, Manfred Nagl et Albert Zündorf. Berlin, Heidelberg : Springer-Verlag. Chap. The Modelling Platform GroIMP and the Programming Language XL, p. 570-572. isbn : 978-3-540-89019-5. doi : 10.1007/978-3-540-89020-1_39. url : http://dx.doi.org/10.1007/978-3-540-89020-1_39.

Koenig, Matthias et al. (2006). « Embedding VTK and ITK into a visual programming and rapid prototyping platform ». In : *Medical Imaging 2006 : Visualization, Image-Guided Procedures, and Display*. T. 6141. International Society for Optics et Photonics, 61412O.

Krueger, Charles W. (1992). « Software reuse ». In : *ACM Computing Surveys* 24.2, p. 131-183.

Kurth, Winfried, Ole Kniemeyerand et Gerhard Buck-Sorlin (2005). « Unconventional Programming Paradigms ». In : *Relational Growth Grammars - A Graph Rewriting Approach to Dynamical Systems with a Dynamical Structure*. Sous la dir. de J.-P. Banatre et al. T. 3566. Springer Berlin / Heidelberg, p. 56-72.

Lattner, Chris et Vikram Adve (2004). « LLVM : A compilation framework for lifelong program analysis & transformation ». In : *Proceedings of the international symposium on Code generation and optimization : feedback-directed and runtime optimization*. IEEE Computer Society, p. 75.

Lecarpentier, Christophe et al. (2019). « WALTer : a three-dimensional wheat model to study competition for light through the prediction of tillering dynamics ». In : *Annals of botany*.

Lee, Edward A (2010). « Disciplined heterogeneous modeling ». In : *International Conference on Model Driven Engineering Languages and Systems*. Springer, p. 273-287.

Lee, Edward A, Stephen Neuendorffer et Michael J Wirthlin (2003). « Actor-oriented design of embedded hardware and software systems ». In : *Journal of circuits, systems, and computers* 12.03, p. 231-260.

Lee, Edward A et Alberto Sangiovanni-Vincentelli (1998). « A framework for comparing models of computation ». In : *IEEE Transactions on computer-aided design of integrated circuits and systems* 17.12, p. 1217-1229.

Li, Mao et al. (2017). « Persistent homology and the branching topologies of plants ». In : *American journal of botany* 104.3, p. 349-353.

Liu, Ji et al. (2015). « A survey of data-intensive scientific workflow management ». In : *Journal of Grid Computing* 13.4, p. 457-493.

Liu, Shouyang et al. (2017). « Estimating wheat green area index from ground-based LiDAR measurement using a 3D canopy structure model ». In : *Agricultural and Forest Meteorology* 247, p. 12-20.

Lobet, Guillaume et al. (2015). « Root system markup language : toward a unified root architecture description language ». In : *Plant Physiology* 167.3, p. 617-627.

Long, Qinqin et al. (2018). « An architecture for the integration of different functional and structural plant models ». In : *Proceedings of the 7th International Conference on Informatics, Environment, Energy and Applications*. ACM, p. 107-113.

Lucas, Mikaël, Laurent Laplaze et Malcolm J. Bennett (fév. 2011). « Plant systems biology : network matters ». In : *Plant, Cell & Environment* 34.4, p. 535-553. issn : 1365-3040. doi : 10.1111/j.1365-3040.2010.02273.x. url : http://dx.doi.org/10.1111/j.1365-3040.2010.02273.x.

Ludäscher, Bertram et al. (2006). « Scientific workflow management and the Kepler system ». In : *Concurrency and Computation : Practice and Experience* 18.10, p. 1039-1065.

Marshall-Colon, Amy et al. (2017a). « Crops In Silico : Generating Virtual Crops Using an Integrative and Multi-scale Modeling Platform ». In : *Front. Plant Sci.* 8. issn : 1664-462X. doi : 10.3389/fpls.2017.00786. url : http://journal.frontiersin.org/article/10.3389/fpls.2017.00786/full.

Marshall-Colon, Amy et al. (2017b). « Crops in silico : generating virtual crops using an integrative and multi-scale modeling platform ». In : *Frontiers in plant science* 8, p. 786.

Martre, Pierre et al. (2018). « The Agricultural Model Exchange Initiative ». In :

McIlroy, M Douglas et al. (1968). « Mass-produced software components ». In : *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, p. 88-98.

Mebatsion, HK et al. (2011). « A novel profile based model for virtual representation of quasi-symmetric plant organs ». In : *Computers and Electronics in Agriculture* 75.1, p. 113-124.

Meyer-Spradow, Jennis et al. (2009). « Voreen : A rapid-prototyping environment for ray-casting-based volume visualizations ». In : *IEEE Computer Graphics and Applications* 29.6, p. 6-13.

Michener, William K et Matthew B Jones (2012). « Ecoinformatics : supporting ecology as a data-intensive science ». In : *Trends in ecology & evolution* 27.2, p. 85-93.

Millar, Andrew J et al. (2019). « Practical steps to digital organism models, from laboratory model species to ‘Crops in silico ». In : *Journal of experimental botany*.

Moulia, Bruno et Meriem Fournier (2009). « The power and control of gravitropic movements in plants : a biomechanical and systems biology view ». In : *Journal of experimental botany* 60.2, p. 461-486.

Oinn, Tom et al. (2004). « Taverna : a tool for the composition and enactment of bioinformatics workflows ». In : *Bioinformatics* 20.17, p. 3045-3054.

Olabarriaga, Silvia D, Tristan Glatard et Piter T de Boer (2010). « A virtual laboratory for medical image analysis ». In : *IEEE Transactions on Information Technology in Biomedicine* 14.4, p. 979-985.

Ong, Yongzhi et al. (2014). « An approach to multiscale modelling with graph grammars ». In : *Annals of botany* 114.4, p. 813-827.

Oppelt, A. L. et al. (2000). « Structure and fractal dimensions of root systems of four co-occurring fruit tree species from *Botswana* ». In : *Annals of Forest Science* 57, p. 463-475.

Pallas, Benoît, Evelyne Costes et Jim Hanan (2016). « Modeling bi-directional signals in complex branching structure : Application to the control of floral induction in apple trees ». In : *Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications (FSPMA), International Conference on*. IEEE, p. 150-157.

Perez, Raphaël PA et al. (2019). « Changes in the vertical distribution of leaf area enhanced light interception efficiency in maize over generations of maize selection ». In : *Plant, cell & environment*.

Pradal, C. et al. (2009). « PlantGL : a Python-based geometric library for 3D plant modelling at different scales ». In : *Graphical Models* 71.1, p. 1-21.

Pradal, Christophe, Gaël Varoquaux et Hans Peter Langtangen (2012). « Publishing scientific software matters ». In : *Journal of Computational Science* 4.5, p. 311-312.

Pradal, Christophe et al. (2008). « OpenAlea : a visual programming and component-based software platform for plant modelling ». In : *Functional Plant Biology* 35.9/10, p. 751-760.

Pradal, Christophe et al. (2015). « OpenAlea : scientific workflows combining data analysis and simulation ». In : *Proceedings of the 27th International Conference on Scientific and Statistical Database Management*. ACM, p. 11.

Pradal, Christophe et al. (2017). « InfraPhenoGrid : a scientific workflow infrastructure for plant phenomics on the grid ». In : *Future Generation Computer Systems* 67, p. 341-353.

Pradal, Christophe et al. (2018). « Distributed management of scientific workflows for high-throughput plant phenotyping ». In : *ERCIM News* 113, p. 36-37.

Prévot, L., F. Aries et P. Monestiez (1991). « Modélisation de la structure géométrique du maïs ». In : *Agronomie* 11, p. 491-503.

Prusinkiewicz, P. et Aristid Lindenmayer (1990). *The Algorithmic Beauty of Plants*. New York, NY, USA : Springer-Verlag New York, Inc.

Prusinkiewicz, Przemyslaw et al. (1999). « L-studio/cpfg : a software system for modeling plants ». In : *International Workshop on Applications of Graph Transformations with Industrial Relevance*. Springer, p. 457-464.

Prusinkiewicz, Przemyslaw et al. (oct. 2009). « Control of bud activation by an auxin transport switch ». In : *Proceedings of the National Academy of Sciences* 106.41, p. 17431-17436. issn : 1091-6490. doi : 10.1073/pnas.0906696106. url : http://dx.doi.org/10.1073/pnas.0906696106.

Puckette, Miller et al. (1996). « Pure Data : another integrated computer music environment ». In : *Proceedings of the second intercollege computer music concerts*, p. 37-41.

Reese, Richard et Dana L Wyatt (1987). « Software reuse and simulation ». In : *Proceedings of the 19th conference on Winter simulation*. ACM, p. 185-192.

Reichstein, Markus et al. (2019). « Deep learning and process understanding for data-driven Earth system science ». In : *Nature* 566.7743, p. 195.

Renton, Michael et al. (2012). « Models of long-distance transport : how is carrier-dependent auxin transport regulated in the stem ? » In : *New Phytologist* 194.3, p. 704-715.

Reyes, Francesco et al. (2018). « MuSCA : a multi-scale model to explore carbon allocation in plants ». In : *bioRxiv*, p. 370189.

Robert, Corinne et al. (2008). « Coupling a 3D virtual wheat (Triticum aestivum) plant model with a Septoria tritici epidemic model (Septo3D) : a new approach to investigate plant-pathogen interactions linked to canopy architecture ». In : *Functional Plant Biology* 35.10, p. 997+. issn : 1445-4408. doi : 10.1071/FP08066. url : http://dx.doi.org/10.1071/FP08066.

Robert, Corinne et al. (2018). « Plant architecture and foliar senescence impact the race between wheat growth and Zymoseptoria tritici epidemics ». In : *Annals of Botany* 121.5, p. 975-989. doi : 10.1093/aob/mcx192. eprint : /oup/backfile/content_public/journal/aob/121/5/10.1093_aob_mcx192/1/mcx192.pdf. url : http://dx.doi.org/10.1093/aob/mcx192.

Sanner, Michel F, Daniel Stoffler et Arthur J Olson (2002). « ViPEr, a visual programming environment for Python ». In : *Proceedings of the 10th International Python conference*, p. 103-115.

Saudreau, Marc et al. (2013). « On the canopy structure manipulation to buffer climate change effects on insect herbivore development ». In : *Trees* 27.1, p. 239-248.

Sievanen, R., A Makela et E Nikinmaa, éd. (1997). *Special issue on Functional-Structural Tree Models*. Helsinki : The Finnish Society of Forest Science.

Sinoquet, H., B. Moulia et R. Bonhomme (1991). « Estimating the three-dimensional geometry of a maize crop as an input of radiation models : comparison between three-dimensional digitizing and plant profiles ». In : *Agricultural and Forest Meteorology* 55, p. 233-249.

Stepanov, Alexander et Meng Lee (1995). *The standard template library*. T. 1501. Hewlett Packard Laboratories 1501 Page Mill Road, Palo Alto, CA 94304.

Strauss, Paul S et Rikk Carey (1992). « An object-oriented 3D graphics toolkit ». In : *ACM SIGGRAPH Computer Graphics*. T. 26. 2. ACM, p. 341-349.

Tardieu, François et al. (2017). « Plant phenomics, from sensors to knowledge ». In : *Current Biology* 27.15, R770-R783.

Ubbens, Jordan et al. (2018). « The use of plant models in deep learning : an application to leaf counting in rosette plants ». In : *Plant Methods* 14.1, p. 6. issn : 1746-4811. doi : 10.1186/s13007-018-0273-z. url : https://doi.org/10.1186/s13007-018-0273-z.

Upson, Craig et al. (1989). « The application visualization system : A computational environment for scientific visualization ». In : *IEEE Computer Graphics and Applications* 9.4, p. 30-42.

Veen, Arthur H (1986). « Dataflow machine architecture ». In : *ACM Computing Surveys (CSUR)* 18.4, p. 365-396.

Vos, J. et al. (mai 2010). « Functional-structural plant modelling : a new versatile tool in crop science ». In : *Journal of Experimental Botany* 61.8, p. 2101-2115. issn : 1460-2431. doi : 10.1093/jxb/erp345. url : http://dx.doi.org/10.1093/jxb/erp345.

Wells, Lisa K et Jeffrey Travis (1997). *LabVIEW for everyone : graphical programming made even easier*. Prentice-Hall.

Weng, Kung-Song (1975). *Stream-oriented computation in recursive data flow schemas*. Massachusetts Institute of Technology. Project MAC.

Whitley, Kirsten N. (1997). « Visual programming languages and the empirical evidence for and against ». In : *Journal of Visual Languages & Computing* 8.1, p. 109-142.

Wilde, Michael et al. (2011). « Swift : A language for distributed parallel scripting ». In : *Parallel Computing* 37.9, p. 633-652.

Yildiz, Ustun, Adnene Guabtni et Anne HH Ngu (2009). « Business versus scientific workflows : A comparative study ». In : *Services-I, 2009 World Conference on*. IEEE, p. 340-343.

Zeigler, Bernard P, Herbert Praehofer et Tag Gon Kim (2000). *Theory of modeling and simulation : integrating discrete event and continuous complex dynamic systems*. Academic press.