



INRAe



LIRMM



 **cirad**

UNIVERSITÉ DE MONTPELLIER
FACULTÉ DES SCIENCES

MASTER 2 INFORMATIQUE
Parcours IMAGINE

Intégration de modèle de simulation L-systems au navigateur de structure morphodynamique MorphoNet.

Rapport de stage

effectué au LIRMM
en collaboration avec l'INRAe (Cirad)
du 01/02 au 01/07 2023
par Aurélien BESNIER

Tuteur en entreprise : M. Emmanuel FAURE, M. Frédéric BOUDON,
M. Christophe GODIN, Mme. Jessica BERTHELOOT
Tuteur à l'Université : Mme. Noura FARAJ

Table des matières

1	Introduction	3
1.1	Introduction	3
1.2	Présentation des organismes d'accueil	3
1.2.1	INRAe	3
1.2.2	LIRMM	3
L'équipe ICAR	4	
1.2.3	CIRAD	4
L'équipe Phenomen	4	
1.3	Problématiques de recherche	5
1.3.1	Présentation du projet Physioscope	5
1.3.2	Problématiques	5
1.3.3	Présentation de la mission	6
2	Environnement technique	7
2.1	Morphonet	7
2.2	L-py	7
2.2.1	L-systèmes	8
2.3	Premier couplage	8
3	Travaux effectués	9
3.1	Amélioration du couplage MorphoNet / L-Py	9
3.1.1	Gestion des erreurs	9
3.1.2	Optimisation de la communication client/serveur	10
Envoie des maillages par paquets de pas de temps	10	
Opérations asynchrones et coroutines pour le chargement des maillages	10	
3.1.3	Exploration des données	12
Gradients	12	
Graphiques	14	
3.2	Calcul de la lumière	16
3.2.1	État de l'art des méthodes de lancer de rayons	16
Ray tracing	16	
Path tracing	18	
Photon Mapping	20	
3.2.2	Implémentation du Photon Mapping	22
3.2.3	Calcul du nombre de photons sur la plante	24
3.2.4	Spectres de la lumière	26

TABLE DES MATIÈRES

Calcul de l'énergie par bandes de spectres	26
Correction du nombre de photons reçu	28
3.3 Modélisation de la chambre de culture	30
3.3.1 Modélisation de la géométrie	30
3.3.2 Matériaux de la chambre	33
Bidirectional scattering distribution function	36
3.3.3 Voisinage des plantes	37
3.3.4 Capteurs virtuels de photons	38
3.4 Résultats	38
4 Conclusion	40
A Glossaire	42
B Annexes	43
Bibliographie	46

Chapitre 1

Introduction

1.1 Introduction

Ce stage de second semestre a eu lieu du 1^{er} février 2023 au 1^{er} juillet 2023 dans le cadre du Master 2, parcours IMAGINE. Il a eu lieu au LIRMM au sein de l'équipe ICAR et au sein de l'institut Agap dans l'équipe Phenomen.

L'objectif de ce stage est de rendre les outils de simulation accessibles aux non-codeurs à travers une interface simple. Le cadre d'étude est la simulation de plantes et leurs conditions d'illuminations. Pour ce faire, la méthode retenue est l'intégration de L-Py au navigateur MorphoNet. Ce stage est la continuation du stage de Mérédith Cérésole en 2022.

1.2 Présentation des organismes d'accueil

1.2.1 INRAe

L'INRAe (Institut National de Recherche pour l'Agriculture, l'alimentation et l'environnement) a été créé en début d'année 2020 à partir de la fusion de l'INRA (Institut National de la Recherche Agronomique) et de l'IRSTEA (Institut National de Recherche en Sciences et Technologies pour l'Environnement et l'Agriculture). Cet institut effectue des recherches autour des domaines de l'agriculture, de l'alimentation et de l'environnement et a pour but d'apporter des solutions pour des enjeux concernant la vie, les humains et la terre. En regroupant des infrastructures de recherche telles que des observatoires, des plateformes et des banques de données, il peut réaliser des améliorations dans différents domaines, comme les sciences de l'eau, les approches à l'échelle des territoires, la conservation et la restauration de la biodiversité, l'anticipation et la gestion des risques, ou l'agriculture numérique.

1.2.2 LIRMM

Le Laboratoire d'Informatique, Robotique et Micro-électronique de Montpellier (LIRMM) est un laboratoire de recherche mixte, dépendant de l'Université de Montpellier et du CNRS. Les travaux principaux de ce laboratoire sont menés par trois départements scientifiques axés sur différents domaines de recherche (informatique, robotique et micro-électronique), eux-mêmes divisés en plusieurs équipes de recherche

L'équipe ICAR

Parmi les équipes du LIRMM, l'équipe ICAR (Image & Interaction), regroupe des chercheurs des deux départements Robotique et Informatique autour de la thématique “image” et, plus généralement, des données visuelles. Cette équipe effectue des recherches sur des thèmes liants interaction et traitement des données visuelles (images 2D, 3D, nD, vidéos ou séquences d’images nD+t, et objets 3D). L’équipe est organisée autour de 4 axes de recherche :

- Analyse & traitement
- Sécurité Multimédia
- Modélisation & Visualisation
- Intelligence Artificielle pour les données visuelles

1.2.3 CIRAD

Le Centre de coopération Internationale en Recherche Agronomique pour le Développement (CIRAD) est l’organisme français de recherche agronomique et de coopération internationale pour le développement durable des régions tropicales et méditerranéennes.

Il contribue, à différentes échelles, à la protection de la biodiversité, aux transitions agroécologiques, à la durabilité des systèmes alimentaires durables, à la santé (des plantes, des animaux et des écosystèmes), au développement durable des territoires ruraux et à leur résistance face au changement climatique.

L'équipe Phenomen

L'équipe PhenoMEn (Phénotypage et Modélisation des plantes dans leur Environnement agro-climatique), fait partie de l'unité mixte de recherche (UMR) AGAP Institut (Amélioration Génétique et Adaptation des Plantes méditerranéennes et tropicales) du Cirad qui se concentre sur les facteurs du développement des plantes et leur adaptation aux contraintes environnementales. L'équipe PhenoMen est l'une des équipes du pôle “Développement et fonctionnement des plantes et des peuplements” de l'institut AGAP, tentant d'identifier et de hiérarchiser les traits et les processus associés expliquant la variabilité des performances des plantes à différents niveaux organisationnels. L'équipe est divisée en axes, collaborant sur ce thème commun. Ces trois axes sont nommés :

- Plant plasticity and ideotype : il s’agit de comprendre les mécanismes d’adaptation des plantes, à des évènements stressants isolés ou récurrents.
- Plant interaction and cropping systems : l’objectif est de valoriser la diversité génétique et les interactions biologiques au sein des agrosystèmes, en analysant différentes options des systèmes de cultures.
- Plant and crop modeling : le but est de contribuer au développement de standards de représentations de données biologiques hétérogènes et de méthodes d’analyses et de simulation de ces données.

1.3 Problématiques de recherche

1.3.1 Présentation du projet Physioscope

Le maintien de la performance des plantes dans des conditions de plus en plus stressantes nécessite de comprendre le réseau physiologique complexe impliqué dans le contrôle du développement de la plante par l'environnement. Du fait de la complexité des mécanismes, l'expérimentation seule n'est pas suffisante, et un outil informatique intégrant un modèle explicitant le fonctionnement physiologique de la plante pourrait s'avérer très pertinent pour aider à l'analyse des expérimentations biologiques. En effet, l'idée est d'intégrer dans le modèle à la fois les connaissances et des hypothèses sur le fonctionnement physiologique, et tester la composante hypothétique en comparant le comportement de la plante *in silico* et le comportement réel. Physioscope vise à développer un tel outil et faire la preuve de concept de son efficacité pour comprendre une problématique biologique spécifique : le contrôle du débourrement des bourgeons par la lumière chez le rosier qui donne naissance à de nouveaux axes sur la plante.

Le projet est subdivisé en 3 axes : (1) Un axe expérimental visant à recueillir des données physiologiques et morphologiques sous différentes conditions expérimentales, (2) un axe "modélisation" visant à créer un modèle intégrant les connaissances sur le fonctionnement physiologique de la plante, (3) un axe informatique visant à développer un outil intuitif, facilement accessible et manipulable par un biologiste. Ce dernier, dans lequel mon stage s'insère, doit pouvoir permettre à l'utilisateur de réaliser facilement des expériences *in silico* et d'autre part d'évaluer les conséquences de ces expérimentations sur la physiologie de la plante et son développement.

Pour cela, L-Py 2.2, un logiciel permet de modéliser le fonctionnement de la plante en interaction avec son développement, est couplé à Morphonet2.1. Sa visée générique et la réponse qu'il apporte à deux verrous scientifiques communs à différentes équipes (interaction fluide avec la plante virtuelle, intégration de processus physiologiques à l'échelle plante) est un levier pour des partenariats internationaux et nationaux, et met en place un continuum entre physiologistes, écophysiologistes, et modélisateurs au sein du métaprogramme Digit-BIO de l'INRAE.

1.3.2 Problématiques

Un des objectifs principaux du projet Physioscope est de rendre accessible des outils de simulations pour des biologistes non avertis en termes de programmation informatique, ou pour des modélisateurs qui n'ont pas développé le code qui les intéresse. Il y a donc un besoin de produire une application simple d'utilisation et fluide pour un public non-informaticien. L-Py permet de faire des simulations de plantes très poussées. Cependant, il faut avoir des connaissances en programmation Python, ce qui limite son utilisation par les personnes extérieures au code. De plus, l'application a besoin d'être facile à installer et de pouvoir fonctionner sur le plus de machines possibles pour faciliter sa diffusion.

De plus, utiliser une plateforme en ligne permet d'éviter les problèmes d'installation compliquée, mais implique d'avoir les complications et problèmes techniques d'avoir une

architecture client/serveur.

Il faut aussi développer des outils dédiés à l'interprétation des résultats expérimentaux obtenus. Ces expérimentations se déroulent en chambre de culture. Dans le cadre de l'étude des plantes, les conditions d'illumination influencent fortement leur développement. Cela nécessite de pouvoir reproduire ces conditions.

1.3.3 Présentation de la mission

Le but de mon stage est de finaliser le couplage de la plateforme MorphoNet avec le simulateur L-Py pour permettre l'exploration et l'interaction avec un modèle de plante virtuelle simulant le réseau physiologique responsable du débourrement des bourgeons axillaires chez le rosier en réponse à différentes conditions d'illumination. Une grosse partie de mon travail s'est focalisée sur le développement d'un modèle de lumière adaptée à l'outil. En plus, j'ai dû assurer une communication avec des biologistes pour définir les caractéristiques intéressantes à développer pour eux.

Chapitre 2

Environnement technique

2.1 Morphonet

MorphoNet[12] est un navigateur interactif de jeux de données segmentés 3D et 3D+t. Il permet de visualiser des données en 3D, mais également de les explorer et de les enrichir. Il permet d'accéder à un ensemble de jeux de données publics ou privés, de données obtenues par observation (microscopique) et de simulations.

Il est développé avec le moteur de jeu Unity. La raison pour laquelle ce moteur est utilisé est qu'il permet de ne pas se soucier (ou peu) sur la partie ingénierie logicielle. Unity permet aussi la portabilité de l'application sur différentes plateformes, Windows, MacOS, Linux et Web.

L'organisation des données se fait par un serveur Python avec l'api Python de MorphoNet. On peut soit utiliser le serveur hébergé sur <https://morphonet.org>, ou une version locale. Pour la version en ligne, il existe une base de données SQL contenant les jeux de données de MorphoNet.

Chaque jeu de données est composé de pas de temps représentant l'état de l'objet à visualiser à un temps t.2.1

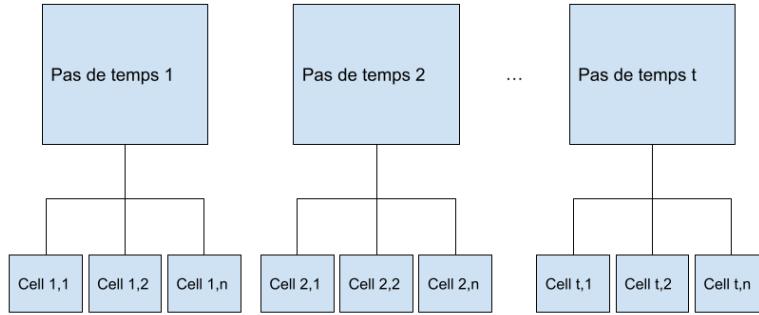


FIGURE 2.1 – Organisation du jeu de données sur l'application

2.2 L-py

L-Py[10] est un logiciel de simulation qui combine la construction des L-systèmes avec le langage de programmation de haut niveau Python. Les L-systèmes ont été conçus comme

un cadre mathématique pour la modélisation de la croissance des plantes. L-Py utilise la bibliothèque de représentations géométriques de données biologiques PlantGL[9].

En plus de ce module logiciel, un environnement de développement visuel intégré a été développé pour faciliter la création de modèles de plantes. En particulier, des outils d'optimisation faciles à utiliser ont été intégrés. Grâce à Python et à son approche modulaire, ce cadre permet d'intégrer une variété d'outils définis dans différents contextes de modélisation, en particulier les outils de la plateforme OpenAlea. De plus, il peut être intégré comme un simple module de simulation de croissance dans des pipelines de calcul plus complexes.

2.2.1 L-systèmes

Un L-système (ou système de Lindenmayer) est une grammaire formelle. Un L-système est défini comme suit :

```
Alphabet: {A, B, C}  
Constantes: {}  
Axiome: A  
Règles: A -> BAC
```

L'alphabet contient l'ensemble des symboles qui représentent différents composants. Ils peuvent être assemblés en chaînes pour créer une structure arborescente. Les constantes sont des paramètres de base du L-système. L'axiome représente l'état initial du L-système. Les règles contrôlent le développement du système. Une application de cet ensemble de règle s'appelle un pas de dérivation. Elles sont appliquées en parallèles sur la chaîne courante à chaque dérivation et, ainsi, définissent l'état initiale de la dérivation suivante.

Dans le cas de L-Py, les L-systèmes permettent de modéliser le processus de développement des plantes.

2.3 Premier couplage

Un premier couplage a été réalisé par Mérédith Cérésole lors d'un stage en 2022. La mission principale de son stage était de permettre de générer une simulation à partir d'un fichier .lp (script de simulation de L-Py), sur MorphoNet, et de pouvoir ensuite manipuler cette simulation générée grâce à différents outils de MorphoNet. De nombreux outils, ainsi que toute l'architecture du projet ont été développés lors de ce stage. L'optique finale était de tester le modèle du rosier sur MorphoNet.

Chapitre 3

Travaux effectués

3.1 Amélioration du couplage MorphoNet / L-Py

Après une communication avec les utilisateurs de l'application, certains problèmes avec le couplage précédent, on était avancés. Un premier problème était celui du temps de chargement initial de la simulation qui était très long (plusieurs minutes) avant de pouvoir interagir avec. Le second était l'absence d'indication des erreurs qui rendaient difficile la maintenance et le développement.

Une première tâche a donc été d'optimiser le travail précédent et de relancer toute l'architecture du projet.

3.1.1 Gestion des erreurs

Avant de l'optimiser, il fallait faire un système de gestion des erreurs. Avant, il fallait se rendre dans les journaux de la machine pour avoir un détail des erreurs.

Pour ce faire, j'ai récupéré les traces d'appels du serveur python pour l'envoyer à l'application client. Cette trace d'appel est ensuite affichée dans une fenêtre "Log" contenant diverses informations de débogage.

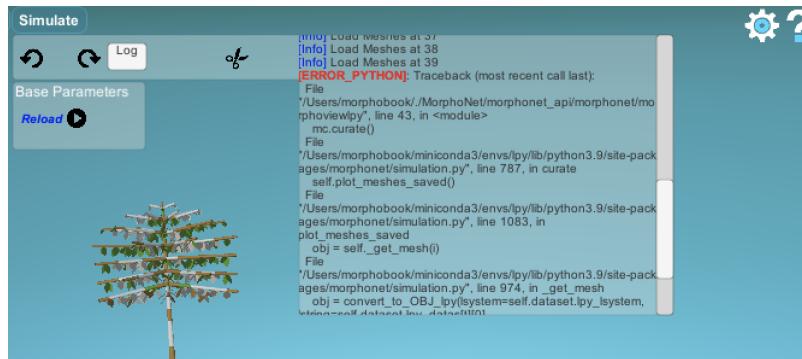


FIGURE 3.1 – Console d'erreur

Cela était simple, mais facilita énormément le travail, car il fallait se rendre dans les journaux du serveur auparavant. De plus, cela m'a permis de me familiariser avec l'organisation du projet.

3.1.2 Optimisation de la communication client/serveur

Une fois avoir la possibilité d'avoir des erreurs claires, il a fallu que j'optimise l'application afin de simplifier l'interaction avec.

Envoie des maillages par paquets de pas de temps

Pour optimiser le chargement initiale de la simulation, j'ai remarqué que chaque pas de temps été envoyé individuellement. Une optimisation simple été donc d'envoyer les pas de temps par paquets pour exploiter au mieux la bande passante et la machine de l'utilisateur pour les importer dans l'application.

Cette optimisation, bien que simple, a permis de réduire le temps de chargement initiale par un facteur de trois.^{3.2}

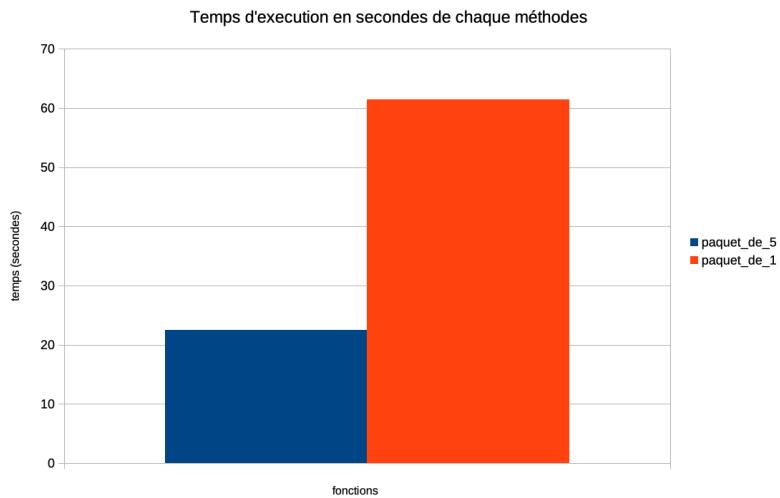


FIGURE 3.2

Opérations asynchrones et coroutines pour le chargement des maillages

Une autre manière d'optimiser une application est de la paralléliser. Une fois l'optimisation précédente réalisée, je me suis intéressé au chargement des maillages dans l'application. Il existe différentes méthodes pour faire cela sur Unity. Une des difficultés principale dans la parallélisation de MorphoNet, est que chaque pas de temps dépend du pas de temps précédents. De ce fait, on ne peut pas paralléliser dans le temps.

La première méthode qui est largement utilisée sur MorphoNet avant mon arrivée est d'utiliser des coroutines¹.

Le plus gros désavantage de cette méthode est que les coroutines fonction sur la "Main Thread" (fil d'exécution principale) du programme Unity, ce qui a pour conséquence de ne

1. <https://docs.unity3d.com/Manual/Coroutines.html>

CHAPITRE 3. TRAVAUX EFFECTUÉS

pas être réellement asynchrone.²

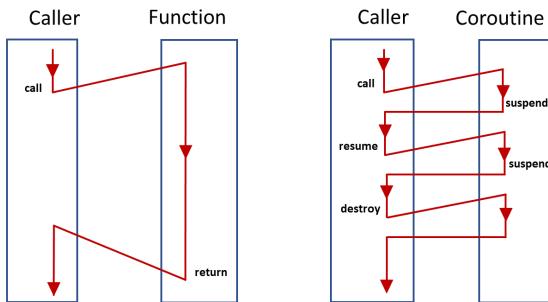


FIGURE 3.3 – Exemple d'exécution d'une coroutine³

La seconde méthode est d'utiliser des fonctions asynchrones C#⁴. Elles permettent, dans Unity, de traiter la logique sans bloquer le fil d'exécution principal, ce qui signifie que les tâches lourdes, ou les tâches qui prennent beaucoup de temps à se terminer, peuvent être exécutées en arrière-plan, pendant que l'application continue à tourner.

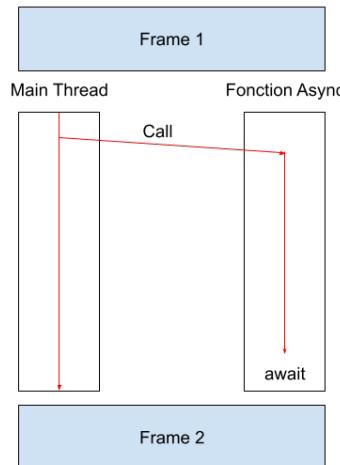


FIGURE 3.4 – Exemple d'exécution d'une fonction asynchrone

Il existe une troisième manière d'améliorer les performances, il s'agit du système "Unity Job"⁵. Cependant, je ne l'ai pas implémenté, car il faudrait changer beaucoup de choses dans MorphoNet.

Ces optimisations ont permis d'accélérer le chargement de la simulation de quelques secondes.

2. <https://docs.unity3d.com/Manual/ExecutionOrder.html>

3. [consulté la dernière fois le 12/06/2023] <https://www.codingninjas.com/codestudio/library/coroutines-in-unity>

4. <https://learn.microsoft.com/fr-fr/dotnet/csharp/language-reference/keywords/async>

5. <https://docs.unity3d.com/Manual/JobSystem.html>

CHAPITRE 3. TRAVAUX EFFECTUÉS

Une fois cette partie d'optimisation réalisée, je me suis intéressé à l'organisation des données entre L-Py et MorphoNet.

3.1.3 Exploration des données

Les simulations L-Py contiennent différentes informations de simulations. Elles peuvent représenter par exemple la concentration en hormone d'un organe, la taille d'une feuille, etc...

Pour explorer ces données, nous avons plusieurs options. Utiliser des gradients de couleurs sur le modèle 3D ou des graphiques pour une exploration plus classiques. Ces informations n'étaient pas très bien visibles sur le modèle, elles ne se mettaient pas bien à jour et la partie de dessin de graphiques n'était pas encore intégrée dans l'application bien qu'il existait une version de teste avec.

Ce fut donc une seconde tâche pour moi, et elle m'a permis de bien apprendre comment les informations entre L-Py et MorphoNet étaient échangées.

Gradients

Ces informations peuvent ensuite être visualisées depuis le menu "Info"3.5. Ce menu permet d'appliquer différentes les cartes de couleur sur le modèle 3D en fonction de ces informations.

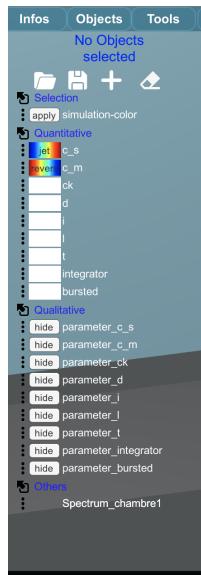


FIGURE 3.5 – Menu "Info" de Morphonet

Ces cartes de couleurs peuvent être appliquées de différentes manières. Pour les simulations, trois modes de visualisation sont à privilégier, à savoir "couleur principale"3.6a,

CHAPITRE 3. TRAVAUX EFFECTUÉS

"couleur secondaire" 3.6b et "halo" 3.6c. À noter que les halos de lumière ne sont pas très visibles selon les conditions d'illumination.

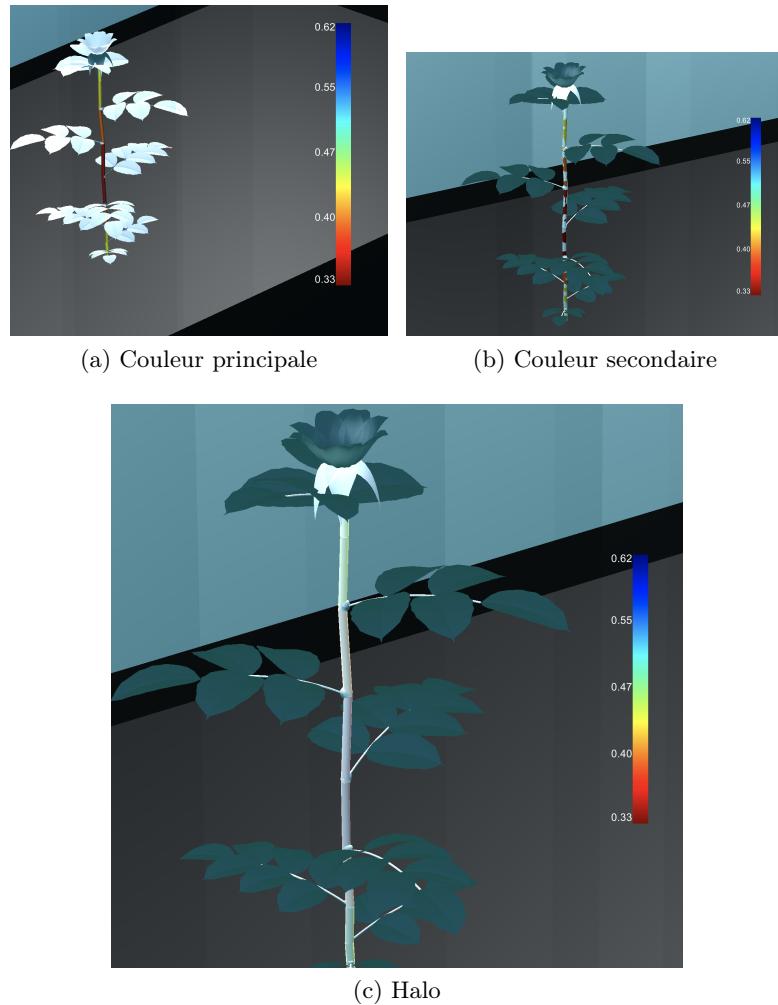


FIGURE 3.6 – Différents gradients de MorphoNet sur la même information

Avec ces gradients de couleur, on peut aussi visualiser deux informations différentes en même temps comme dans la figure 3.7.

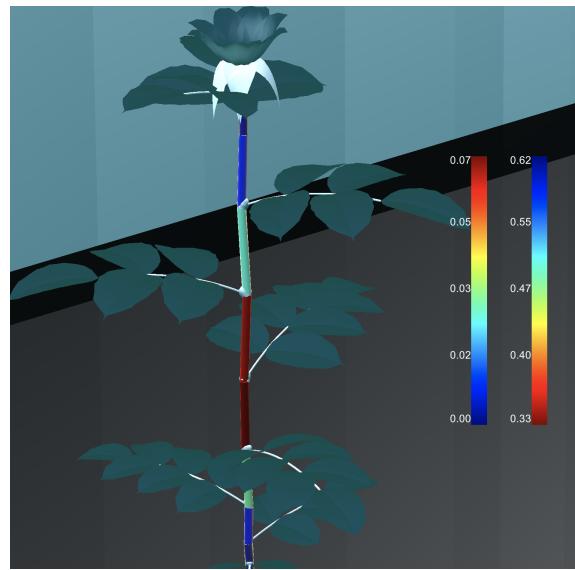


FIGURE 3.7 – Visualisation de deux informations différentes avec deux cartes de couleur de différent types

Graphiques

L'interface permettant de faire des graphiques se situe dans le menu "Graph" 3.8. Ce menu permet de dessiner les graphiques des différentes informations de la simulation au cours du temps et de les visualiser directement dans l'application 3.9a ou de les sauvegarder dans un fichier CSV ou sous forme d'image png 3.9b.

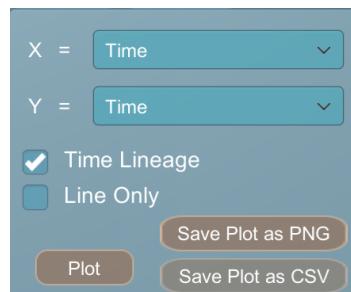


FIGURE 3.8 – Menu Graph

CHAPITRE 3. TRAVAUX EFFECTUÉS

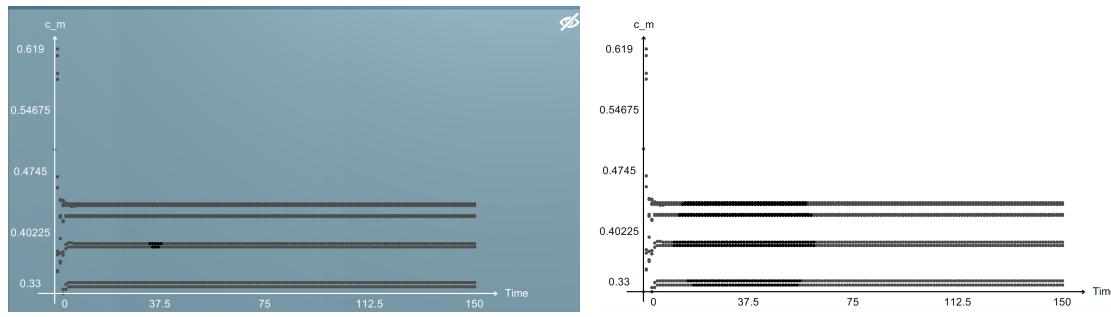


FIGURE 3.9 – Exemple de graphique

Ces graphiques sont aussi affectés par les cartes de couleurs sélectionnées du menu "Info".3.10

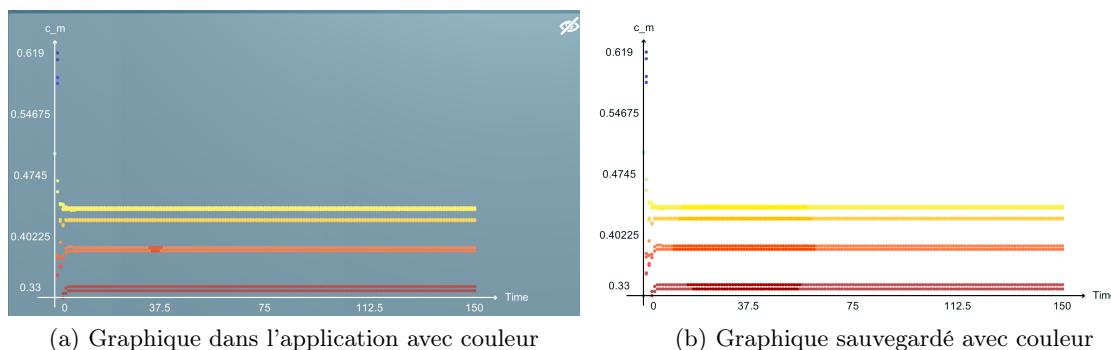


FIGURE 3.10 – Exemple de graphique

Ce menu permet aussi avec la case à cocher "Time Lineage"3.8 de dessiner le graphique des organes sélectionnés 3.11. Il existe aussi la possibilité de cocher "Line Only" pour dessiner le graph avec des lignes plutôt que des cercles.

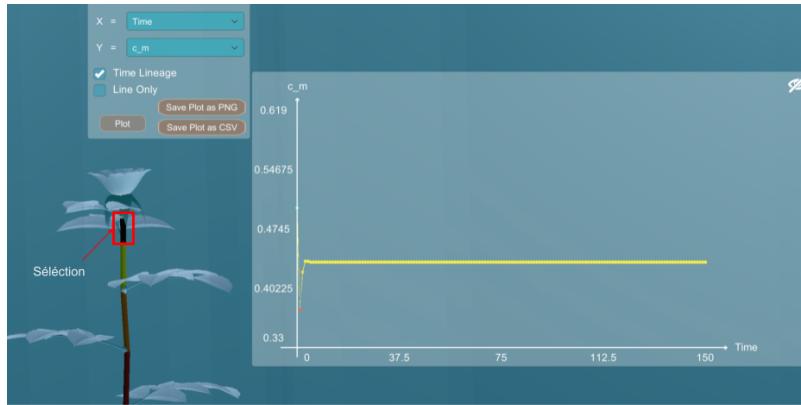


FIGURE 3.11 – Exemple de graphique d'un organe sélectionné

3.2 Calcul de la lumière

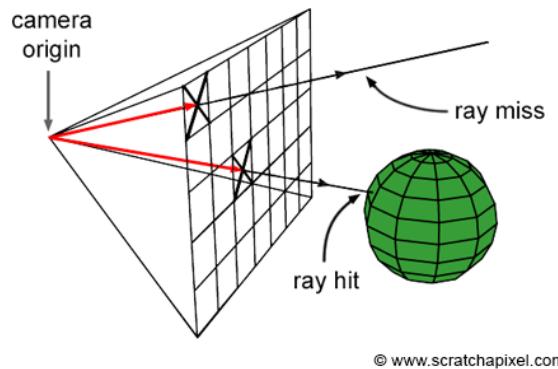
La première intuition que j'ai eue pour la simulation de la lumière était d'utiliser le ray tracing. Cependant, les méthodes de ray tracing classiques ne semblaient pas appropriées, car le temps d'exécution était trop long et les résultats dépendent de la position d'une caméra. Malgré ce premier constat, une méthode s'est distinguée, le photon mapping comme détaillé dans la partie 3.2.1.

3.2.1 État de l'art des méthodes de lancer de rayons

Pour justifier ce choix, je vais tout d'abord comparer différentes méthodes de lancer de rayons les plus communes.

Ray tracing

Le ray tracing est un terme générique qui recouvre les techniques de rendu qui trace des rayons entre une scène et une caméra 3.12. Le ray tracing a été pour la première fois proposé pour le rendu d'image par ordinateur par Arthur Appel en 1968 [1]. En utilisant la visibilité primaire, Appel a utilisé le ray tracing pour déterminer la surface la plus proche de la caméra à chaque point de l'image. Pour déterminer si un point est dans l'ombre ou non, il a suivi les rayons secondaires depuis chaque point ombragé jusqu'à la source de lumière.



© www.scratchapixel.com

FIGURE 3.12 – Principe de base du ray tracing⁶

Généralement, lorsque l'on parle de ray tracing de nos jours sans préciser la méthode, on se réfère à la méthode de Turner Whitted[2]. Cette méthode permet aussi de gérer les effets de miroirs et les effets de réfraction de la lumière.

Pour réaliser ces effets, on procède ainsi. Lorsqu'un rayon lancé depuis la caméra touche une surface miroir, il rebondit jusqu'à toucher une surface diffuse où se détruit après un certain nombre de rebonds. Une fois cette surface touchée, on détermine l'illumination du point d'intersection par rapport aux sources de lumière de la scène 3.13a puis on revient sur le chemin inverse jusqu'à la caméra 3.13b.

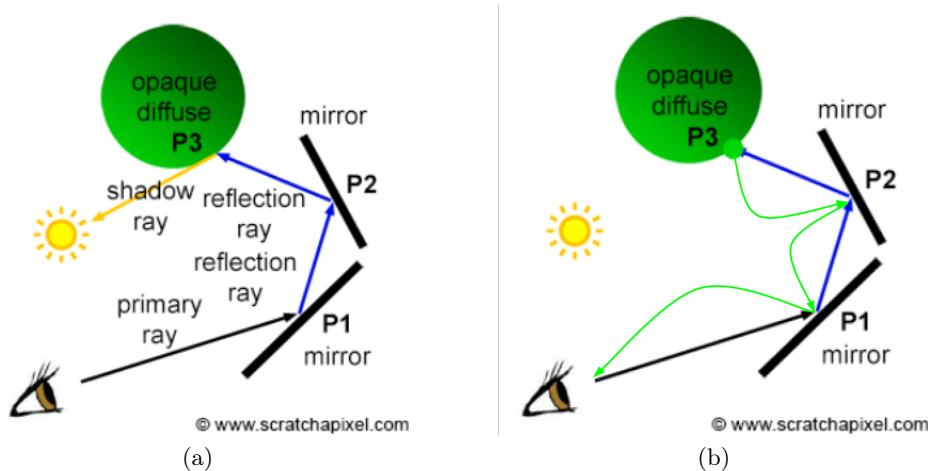


FIGURE 3.13 – Méthode de réflexion de Whitted

Pour les effets de réfraction, on procède de la même manière, mais au lieu d'une réflexion, on effectue une réfraction dans le milieu touché 3.22.

6. [consulté la dernière fois le 23/06/2023] : <https://www.scratchapixel.com/>

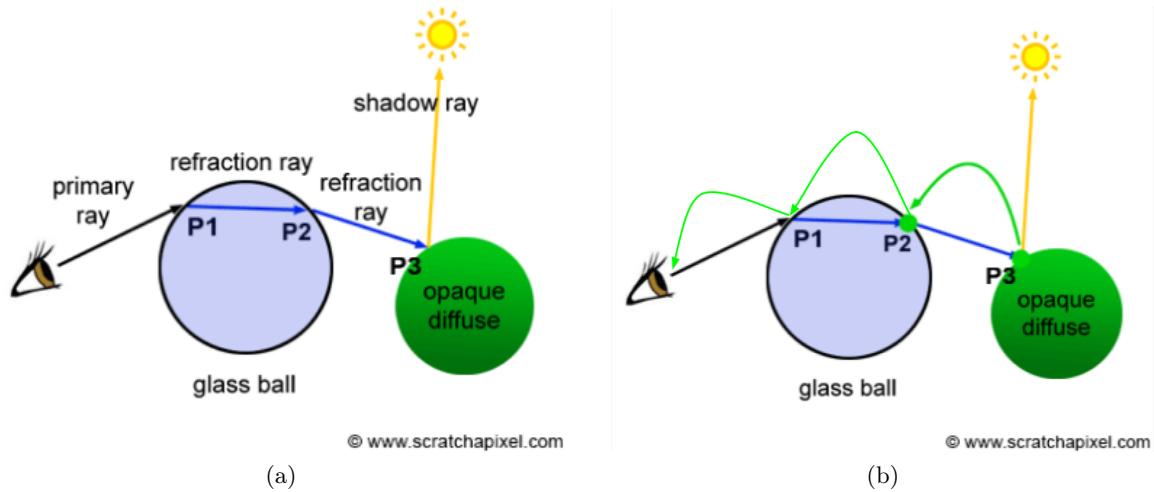


FIGURE 3.14 – Méthode de réfraction de Whitted

Cette méthode permet d'avoir des rendus comme dans la figure 3.15

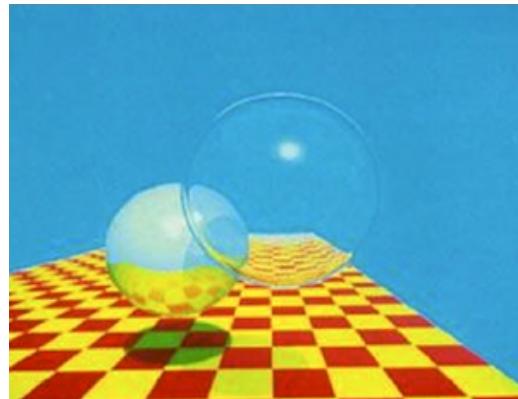


FIGURE 3.15 – Exemple de rendu dans le papier de Whitted

Cette méthode, bien que capable de produire des images avec une illumination correcte, est très demandeuse en calcul. En effet, pour obtenir une image ou l'illumination fidèle à la réalité, il faut un grand nombre d'échantillons par pixel de l'image, ce qui implique aussi d'avoir beaucoup de rayons à lancer et entraîne donc des problèmes de mémoire. Elle est aussi dépendante de la vue d'une caméra. C'est pourquoi je ne l'ai pas retenue.

Path tracing

Le path tracing et le ray tracing sont tous deux des algorithmes de rendu d'image qui trace le chemin de la lumière à travers une scène 3D. La différence entre les deux est la méthode de résolution de ce chemin.

CHAPITRE 3. TRAVAUX EFFECTUÉS

Le path tracing est un algorithme qui vise à résoudre l'équation de rendu posé par James T. Kajiya[3]. Cette équation est la suivante :

$$I(x, x') = g(x, x')[\epsilon(x, x') + \int_s \rho(x, x', x'') \Gamma(x', x'') dx''] \quad (3.1)$$

où :

- $I(x, x')$ est liée à l'intensité de la lumière passant du point x' au point x .
- $g(x, x')$ est un terme de "géométrie".
- $\epsilon(x, x')$ liée à l'intensité de la lumière émise de x' à x .
- $\rho(x, x', x'')$ est liée à l'intensité de la lumière diffusée de x'' à x par une zone de la surface en x' .

Avec cette équation, Kajiya a montré que le problème de la puissance de traitement limitée, en travaillant sur un nombre toujours croissant de rayons, pouvait être résolu par l'utilisation d'un échantillonnage statistique de la scène.

Le ray tracing classique consiste à calculer la trajectoire exacte de réflexion ou de réfraction de chaque rayon et à les retracer jusqu'à une ou plusieurs sources de lumière. Avec le path tracing, plusieurs rayons sont générés pour chaque pixel, mais ils rebondissent dans une direction aléatoire. Ce procédé se répète lorsqu'un rayon touche un objet et se poursuit jusqu'à ce qu'une source lumineuse soit atteinte, ou qu'une limite de rebond pré définie soit atteinte.

Tous les rayons ne seront pas utilisés pour créer la couleur finale du pixel dans l'image 3.16. Le path tracing permet d'obtenir une trajectoire presque idéale des rebonds de la lumière, de la caméra à la source lumineuse. Pour améliorer la précision de l'image, il est possible de modifier le nombre d'échantillons pour chaque pixel, de manière semblable au ray tracing.

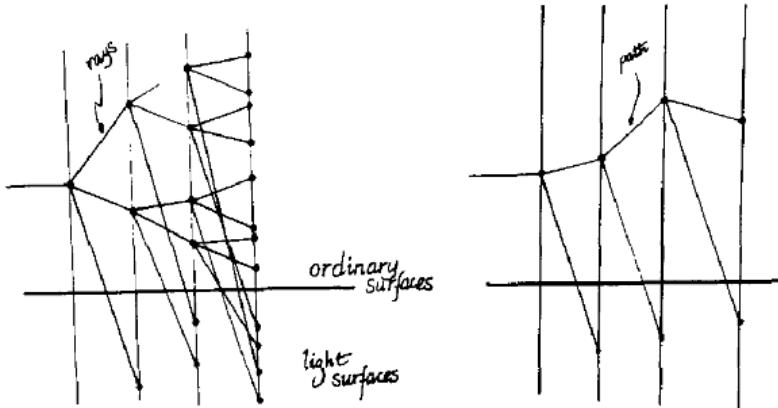


FIGURE 3.16 – Diagramme de Kajiya[3] pour démontrer la réduction du nombre de rayons

Cette méthode permet d'obtenir des rendus similaires, mais avec une meilleure illumination 3.17 pour moins de calcul. Cependant, encore une fois, elle dépend d'un point de vue particulier dans la scène. Il a fallu que je trouve une autre méthode.

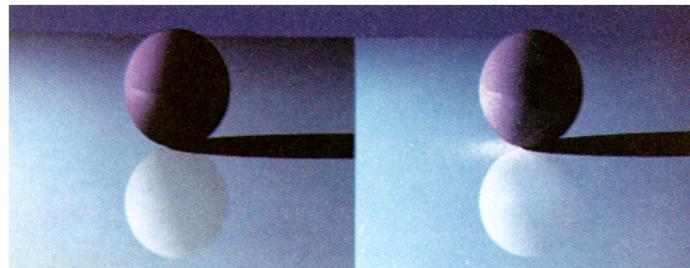


FIGURE 3.17 – Exemple de rendu avec le path tracing de Kajiya[3], ray tracing classique à gauche, path tracing à droite

Photon Mapping

Le photon mapping est, lui aussi, un algorithme de rendu d'image. Il a été proposé par Henrik Wann Jensen[7] en 1996. La grande différence avec les deux algorithmes précédents est qu'il se divise en deux phases : le photon scattering/tracing et le photon gathering/collecting. Ces termes varient dans la littérature.

La première étape consiste à envoyer des photons dans la scène 3D 3.18. Un photon peut être représenté par la structure suivante⁷ :

```
struct photon {
    float x, y, z;           // position ( 3 x 32 bit floats )
    char p[4];               // power(rgb) packed as 4 chars
    char phi, theta;         // compressed incident direction
    short flag;              // flag used for kd-tree
}
```

Ces photons sont ensuite stockés dans une structure de données connue sous le nom de photon map (carte de photons). Selon l'implémentation, il peut y avoir une ou deux photon map. Si l'on utilise deux photon maps, la première est une carte caustique à haute définition qui permet de visualiser directement les objets d'une scène, et la seconde est une estimation grossière à basse résolution qui est utilisée dans la phase de rendu de la méthode.

La seconde étape consiste à envoyer des rayons depuis la caméra et de se référer à la photon map calculé auparavant pour pouvoir déterminer l'illumination d'un point 3.18. Cette formule d'illumination est l'équation de rendu [3], la même que pour le path tracing.

7. [consulté la dernière fois le 14/02/2023] https://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/photon_mapping/PhotonMapping.html

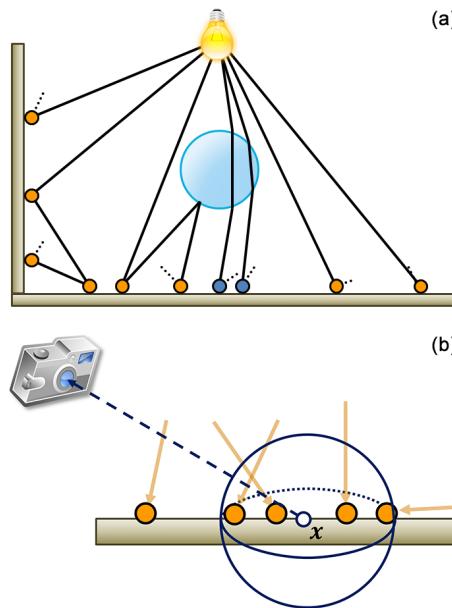


FIGURE 3.18 – Photon mapping, (a) Photon Scattering, (b) Photon Gathering[11]

Après avoir testé quelques projets de photon mapping disponible sur le web, j'ai remarqué que la phase de calcul de la photon map était très rapide. Alors, j'ai eu l'idée d'utiliser cette photon map pour calculer l'énergie reçue sur chaque objet de la scène.

Le principe est le suivant, lorsqu'un photon touche un élément de la scène à un instant t on récupère sa valeur d'énergie et on l'ajoute à un objet. Ainsi, on peut calculer la quantité de lumière que reçoit chaque objet. Aussi, cette première étape ne dépend pas de la position d'une caméra, ce qui peut permettre de faire quelque chose de générique en fonction de l'environnement que peut choisir l'utilisateur.

De plus, le photon mapping permet aussi de calculer les caustiques. Les caustiques désignent en optique et en mathématiques l'enveloppe des rayons lumineux subissant une réflexion ou une réfraction sur une surface ou une courbe. Ceux-ci peuvent être utilisés lors ce que l'on veut simuler l'évolution d'une plante dans un milieu non naturel (serre, bocal, etc...).

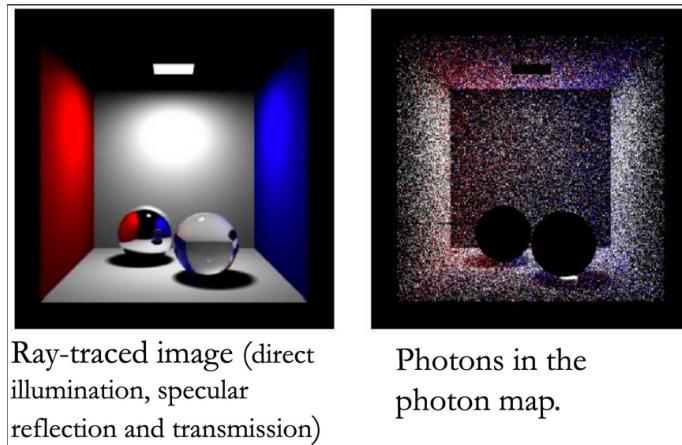


FIGURE 3.19 – Représentation de la "photon map" avec un rendu par ray tracing⁸

3.2.2 Implémentation du Photon Mapping

Mon implémentation du photon mapping se base sur un projet GitHub⁹. L'une des raisons principales d'avoir choisi ce projet comme point de départ est qu'il est codé en C++. Cela permet d'assurer des bonnes performances et une utilisation des ressources optimale étant donné que le calcul de lumière est notoire pour utiliser des ressources abondantes.

Seulement, pour pouvoir interagir avec le serveur de MorphoNet qui est en Python, j'ai dû faire un "wrapper" Python. Ce "wrapper" a été fait à l'aide de pybind11¹⁰.

Cette implémentation dispose de plusieurs avantages. Dans un premier temps, elle est minimale ce qui permet de l'étendre assez facilement. Un autre avantage est qu'elle se base sur la bibliothèque de ray tracing Embree¹¹. Cette bibliothèque de calculer de manière optimisée les intersections Rayons/triangles de scènes 3D. Il existe d'autres bibliothèques de calcul dédiées au ray tracing comme OptiX¹² de NVIDIA ou encore RadeonRays¹³ d'AMD, cependant ces bibliothèques reposent sur l'utilisation des cartes graphiques. Ce ne serait pas un désavantage en temps normal, mais dans une architecture client-serveur comme la nôtre, l'accès aux ressources GPU peuvent poser problèmes si deux utilisateurs utilisent l'outil au même moment.

L'intégration de ce projet au sein du couplage L-Py / MorphoNet se fait comme suit :

8. [consulté la dernière fois le 23/06/2023]<https://www3.cs.stonybrook.edu/~qin/courses/visualization/visualization-radiosity-and-photon-mapping.pdf>

9. https://github.com/yumcyaWiz/photon_mapping

10. <https://pybind11.readthedocs.io/en/stable/>

11. <https://www.embree.org/>

12. <https://developer.nvidia.com/rtx/ray-tracing/optix>

13. <https://gpuopen.com/radeon-rays/>

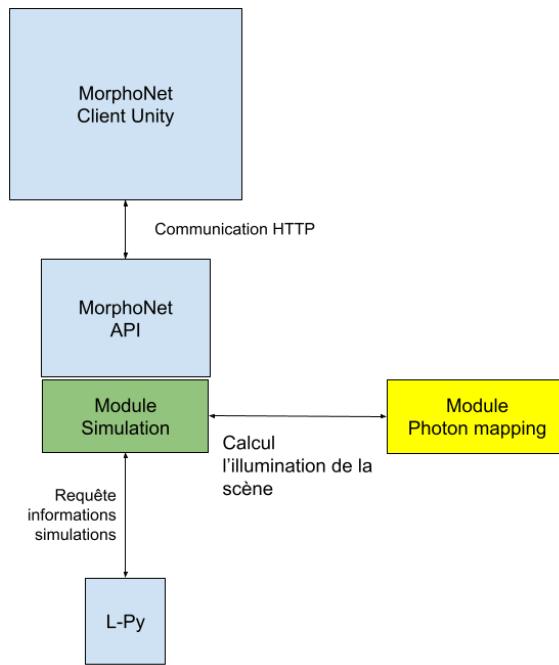


FIGURE 3.20 – Intégration du module de Photon Mapping à l’API de MorphoNet

Pour tester si cela fonctionnait, j’ai effectué quelques rendus en envoyant la scène depuis MorphoNet au module de photon mapping.



FIGURE 3.21 – Photon map obtenu à partir de la scène envoyée depuis le client de MorphoNet

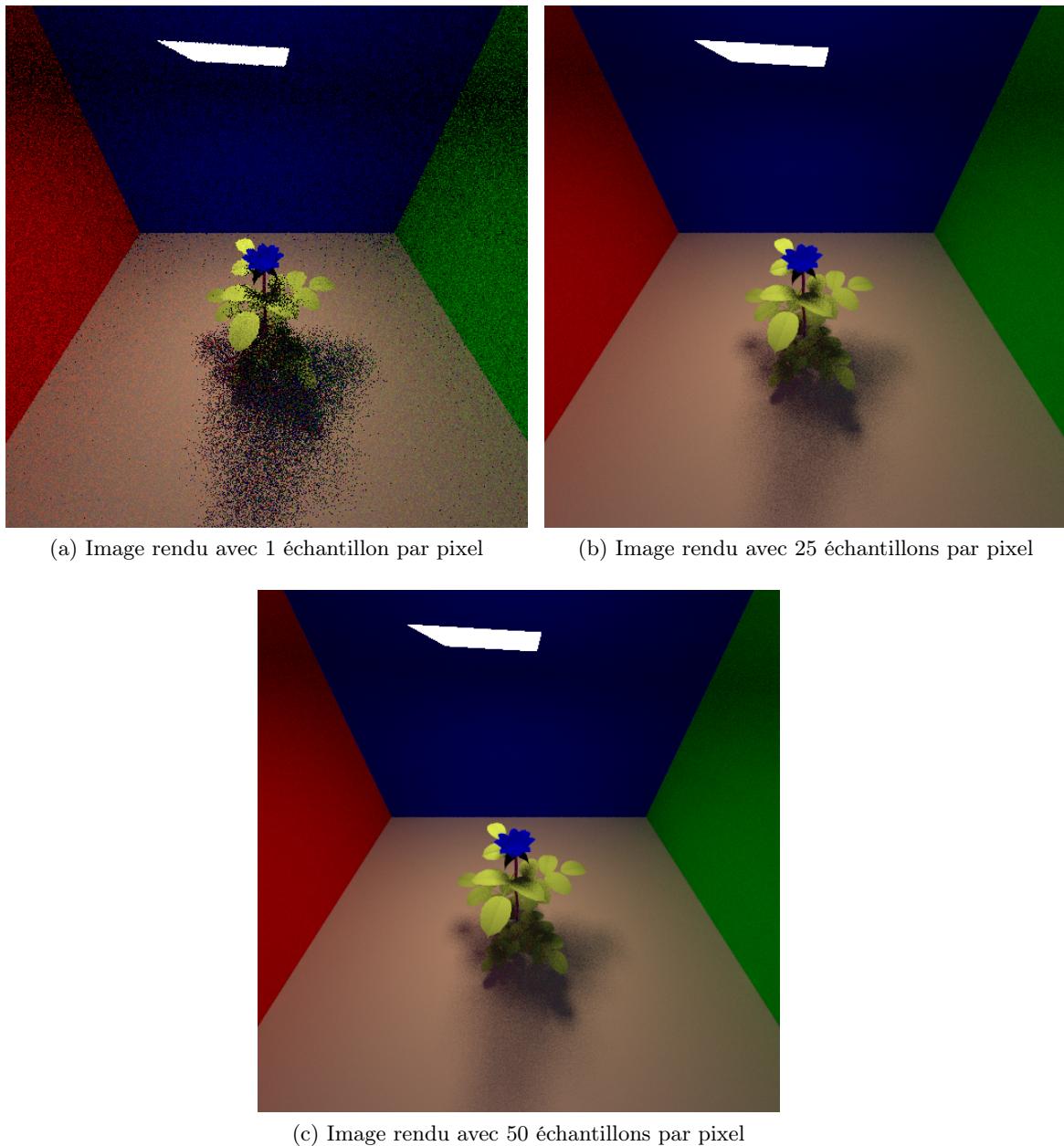


FIGURE 3.22 – Différences d’images rendues selon le nombre d’échantillons

3.2.3 Calcul du nombre de photons sur la plante

Pour calculer la quantité de photons sur chaque organe de la plante, on parcourt la photonmap et l’on regarde l’identifiant du triangle touché pour savoir s’il appartient à un organe du maillage L-Py. Si oui, on incrémente un compteur dans un dictionnaire avec

comme clé l'identifiant d'organe de la simulation L-Py.1

Algorithm 1 : compute_energy

Données : shape_energy : dictionnaire, photonmap : PhotonMap,
triangle2shape : dictionnaire

Début

```

pour tous photon dans photonmap faire
    triId ← triangle2shape.get(photon.triId);
    si triId n'est pas nul alors
        //Alors on a touché un organe de la simulation.
        si triId est dans shape_energy alors
            | shape_energy[triId] ← shape_energy[triId] + 1;
        finsi
        sinon
            | shape_energy[triId] ← 1;
        finsi
    finsi
finptrs
Fin

```

Une fois cette information créée, on l'envoie au client. Voici un exemple de résultat 3.23. L'information est accessible dans le menu "Info" et l'on peut y appliquer une échelle de couleur comme pour les autres informations. On peut aussi la visualiser sous forme de graphique 3.24. Cette information est calculée à partir du pas de temps sélectionné jusqu'à la fin de la simulation.

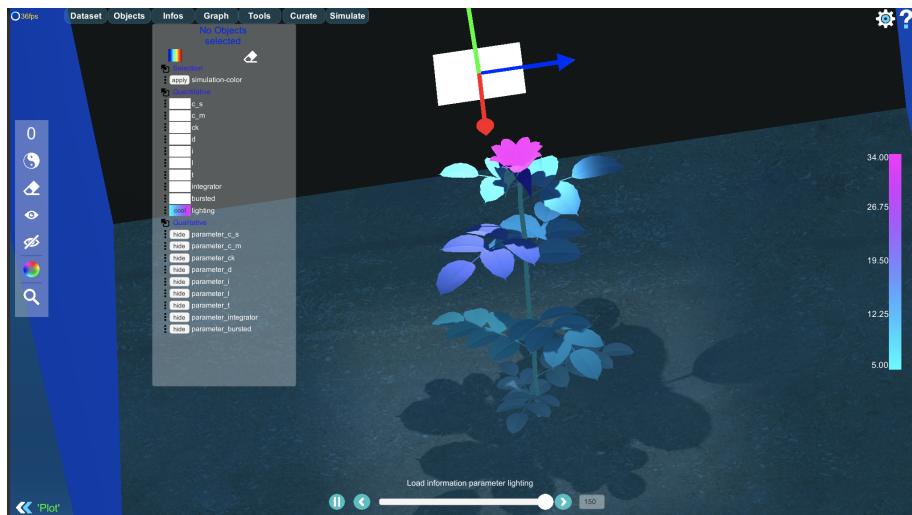


FIGURE 3.23 – Résultat de l'information de calcul de photons sur chaque organe.

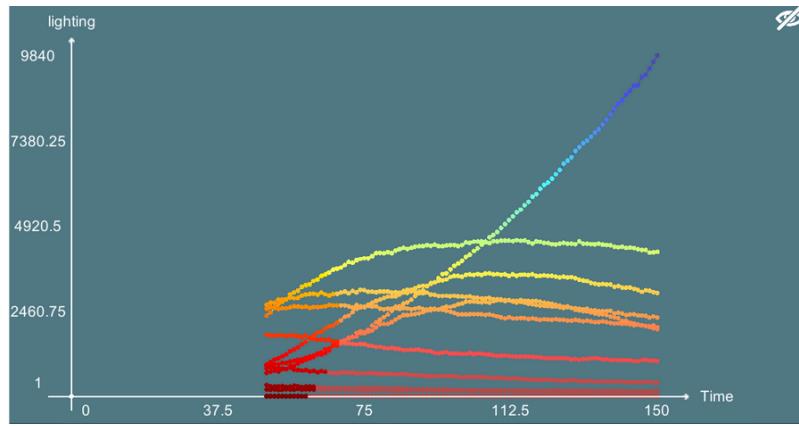


FIGURE 3.24 – Résultat de l’information en graphique.

3.2.4 Spectres de la lumière

Calcul de l’énergie par bandes de spectres

Pour calculer l’énergie lumineuse reçue sur chaque organe de la plante, nous avons divisé les spectres de lumière en plusieurs bandes. Les bandes par défaut sont présentées dans la table 3.1 et la figure 3.25. Il est également possible pour l’utilisateur de personnaliser les bandes sur lesquelles ils souhaitent faire les simulations de lumière (Voir figure 3.26).

Nom de bande	Longueur d’ondes
Bleue	400-500 nm
Vert	500-600 nm
Rouge	600-700 nm
Rouge profond	700-800 nm
PAR	400-700 nm

TABLE 3.1 – Décompositions utilisées

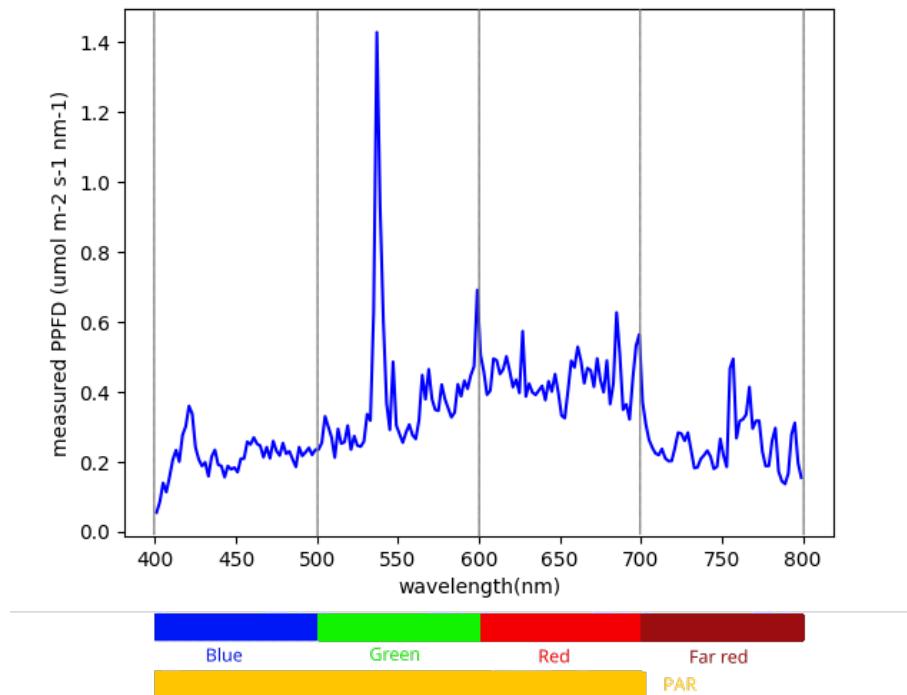


FIGURE 3.25 – Exemple de décomposition d'un spectre de lumière

L'utilisation de ces bandes se fait comme suit. L'utilisateur sélectionne une bande dans laquelle il souhaite simuler l'éclairage, ainsi que le nombre de photons qu'il souhaite envoyer dans cette bande. À partir de ces informations, la simulation se fera à partir de la longueur d'onde qui correspond le plus à la moyenne de la bande de spectre sélectionnée. Par exemple, avec le spectre de la figure 3.25 si l'utilisateur sélectionne la bande bleue, la simulation se fera sur la longueur d'onde de 435 nanomètres.

Pour le PAR (Photosynthetically active radiation), qui désigne la gamme spectrale du rayonnement solaire de 400 à 700 nanomètres que les organismes photosynthétiques sont capables d'utiliser dans le processus de photosynthèse, le calcul est un peu différent des autres bandes. On calcule l'énergie des bandes de 100 nanomètres et l'on en fait la somme. Cela revient à calculer la bande Bleue + Verte + Rouge.

Il existe aussi la possibilité de faire des bandes sur mesure par l'utilisateur à partir de l'interface 3.26. On y sélectionne une longueur d'onde inférieure et une supérieure ainsi qu'un intervalle d'échantillonnage. Lorsqu'une bande sur mesure dépasse 100 nanomètres, on subdivise cette bande selon les bandes prédéfinies sur les bandes prédéfinies ci-dessus 3.1 pour prendre en compte les gammes de longueur d'onde auxquelles la plante est sensible.

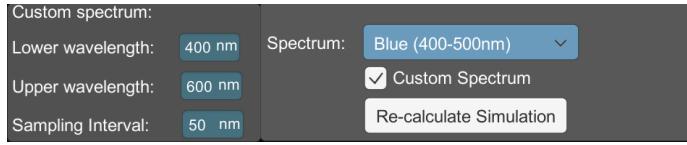


FIGURE 3.26 – Interface de sélection de bande du spectre

Correction du nombre de photons reçu

Les lampes des chambres de culture n'émettant pas le même nombre de photons selon les longueurs d'ondes étudiées en réalité (non-uniformité du spectre, voir figure 3.25), il a fallu trouver un moyen d'adapter les simulations à la longueur d'onde. Pour simplifier l'utilisation sur mon interface, je demande à l'utilisateur de rentrer un seul nombre de photons à lancer. Le nombre de photons indique le niveau de précision du calcul.

Pour éviter d'avoir un nombre de photons trop bas dans certaines longueurs d'onde (là où l'émission de la lampe est faible), nous avons choisi d'appliquer cette correction a posteriori, c'est-à-dire une fois les photons lancés, appliquer la correction sur le nombre de photons obtenu sur chaque organe. Si une bande dispose d'un nombre de photons à envoyer trop faible, il est fortement possible qu'aucun photon de cette bande ne touche la plante.

Les calculs suivants ne fonctionnent que pour des lampes de même type, en effet, une entrée du modèle est la puissance des lampes, et l'on sait que les LED par exemple sont moins consommatrices d'énergie que les autres et donc une puissance moindre pour une intensité lumineuse similaire à d'autres types de lampes.

Entrées de la simulation de lumière :

Soit N lampes, avec leur puissance l (watt) : $P^{tot}(l)$

Soit le spectre de la lampe, donnant pour chaque longueur d'onde λ (un entier, en nm), une valeur du flux de photons émis ($mol\ photons\ s^{-1}$) : $PF^a(l, \lambda)$

On normalise le flux de photons sur chaque λ pour obtenir le flux de photons relatif $PF^r(l, \lambda)$ tel que :

$$PF^r(l, \lambda) = \frac{PF^a(l, \lambda)}{\sum_{li=300}^{800} PF^a(l, li)} \quad (3.2)$$

Flux de photon de la lampe l ($mol\ photons\ s^{-1}$), $PF(l)$, déduite de la puissance :

$$PF(l) = \frac{P^{tot}(l)}{\sum_{\lambda=300}^{800} PF^r(l, \lambda) \times E^{ph}(\lambda)} \quad (3.3)$$

où

$$E^{ph}(\lambda) = h \times f(\lambda) = \frac{h \times c}{\lambda \times 10^{-9}} \quad (3.4)$$

Avec E^{ph} en joules, h la constante de Planck (6.626×10^{-34} J s), c la vitesse de la lumière (3×10^8 m s $^{-1}$), λ la longueur d'onde en m.

Avec la subdivision du spectre en plusieurs bandes i telles que : $a(i) < \lambda(i) \leq b(i)$

Corrections appliquées en sortie de la simulation :

Soit le nombre de photons, issus de la lampe l , reçus par chaque élément e de la scène par bande i : $N_e^{ph}(l, i)$.

Les calculs suivants ne peuvent être faits que si l'on fait une simulation par lampe à la fois. On applique les deux corrections suivantes :

- Correction C_e^{spec} pour prendre en compte l'hétérogénéité d'intensité selon la longueur d'onde pour une lampe l :

$$C_e^{spec}(l, i) = \sum_{\lambda=a(i)+1}^{b(i)} PF^r(l, \lambda) \quad (3.5)$$

- Correction C_e^{int} pour prendre en compte les intensités potentiellement différentes des différentes lampes l :

$$C_e^{int}(l) = \frac{PF(l)}{\sum_{l=n}^N PF(n)} \times N \quad (3.6)$$

L'application de ces corrections se fait ainsi :

$$N_e^{ph,corr}(l, i) = N_e^{ph}(l, i) \times C_e^{spec}(l, i) \times C_e^{int}(l) \quad (3.7)$$

L'implémentation de ces corrections sous forme algorithmique pour l'énergie donnée par une lampe peut se faire de cette manière 2 :

Algorithme 2 : correct_energy

Données : bande : dictionnaire, spectre_entier : dictionnaire,
 shape_energy_lamp : dictionnaire, correction_ratio : flottant,
 lambda : entier, puissance : flottant, nb_photons : entier, pf_tot :
 flottant, nb_lampes : entier

Début

```

    pfr_sum ← 0;
    pour tous lambda, pfa de bande faire
        pfa_sum ← 0;
        pour tous cle, valeur de spectre_entier faire
            | pfa_sum ← pfa_sum + valeur;
            finptrs
        pfr_sum ← pfa/pfa_sum;
        finptrs
        Cspec ← pfr_sum;
        e_photon ← (h × c)/(lambda * 10-9);
        pf ← puissance/pfr_sum × e_photon;
        Cint ← pf/pf_tot × nb_lampes;
        correction_energy ← nb_photons × Cspec × Cint;
        pour tous cle, valeur de shape_energy faire
            | shape_energy[cle] ← valeur × correction_energy;
        finptrs
    Fin

```

3.3 Modélisation de la chambre de culture

Pour pouvoir simuler de manière fidèle l'illumination de la plante, il faut aussi modéliser l'environnement dans lequel elle évolue.

3.3.1 Modélisation de la géométrie

Pour modéliser la chambre de culture, je me suis basé sur des informations que m'ont fournies les biologistes et sur un ancien projet intitulé SEC2[8].

Les chambres, dans ce projet, étaient modélisées avec des fichiers .rad[4][6]. Cependant, ce format étant daté et peu utilisé, j'ai préféré les convertir en format .obj¹⁴ qui est plus standard dans le monde de la 3D. Il existe aussi plus de logiciel capable de lire ou créer ce format d'objets 3D, ce qui permet d'assurer l'évolution de l'outil.

La configuration de la chambre sur laquelle nous voulons valider les simulations de lumière est faite comme suit ?? avec une première représentation virtuelle ?? :

14. [https://fr.wikipedia.org/wiki/Objet_3D_\(format_de_fichier\)](https://fr.wikipedia.org/wiki/Objet_3D_(format_de_fichier))

CHAPITRE 3. TRAVAUX EFFECTUÉS

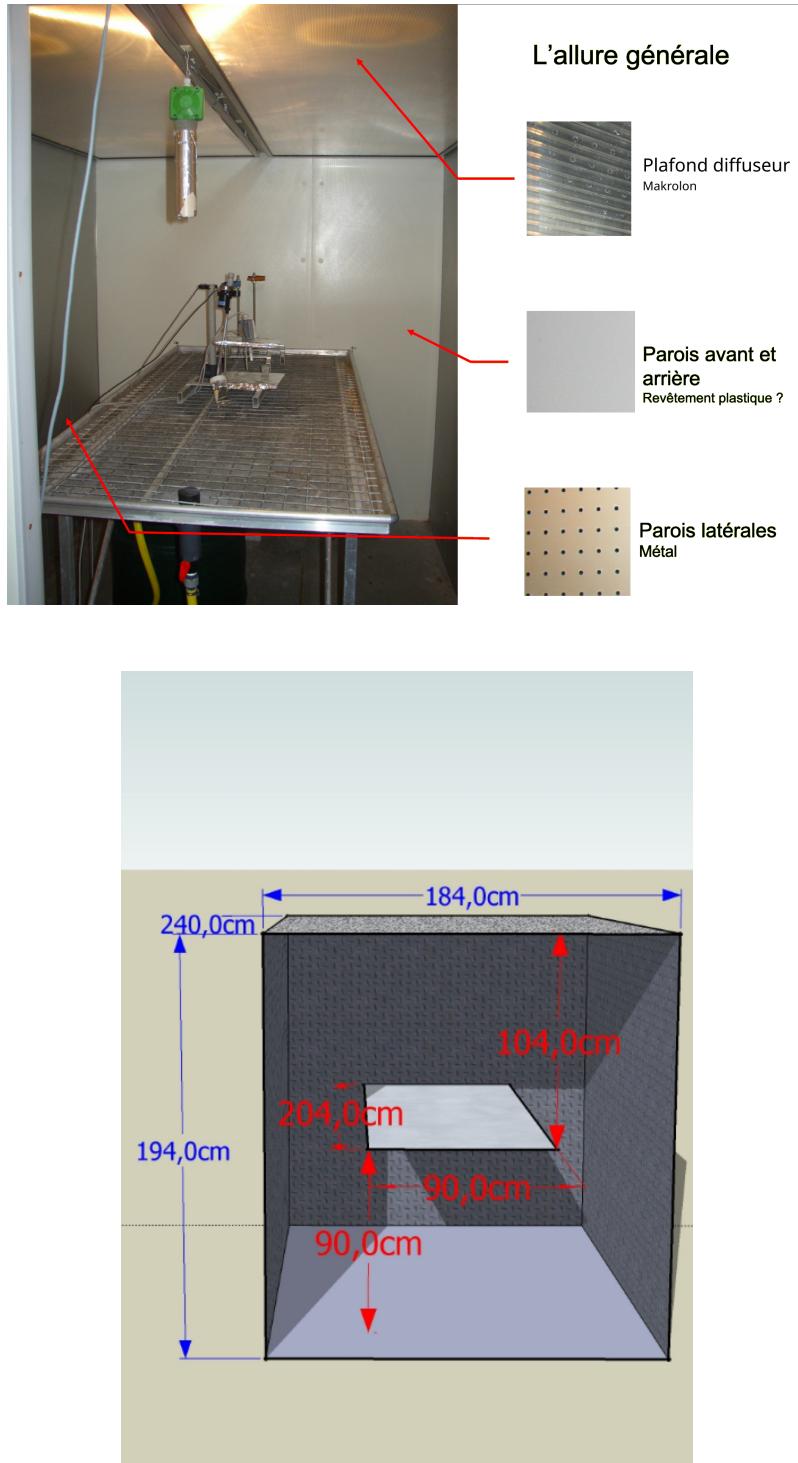


FIGURE 3.27 – Représentation virtuelle de la disposition

CHAPITRE 3. TRAVAUX EFFECTUÉS

Les lampes sont disposées dans quatre quadrants identiques au niveau du plafond 3.28.

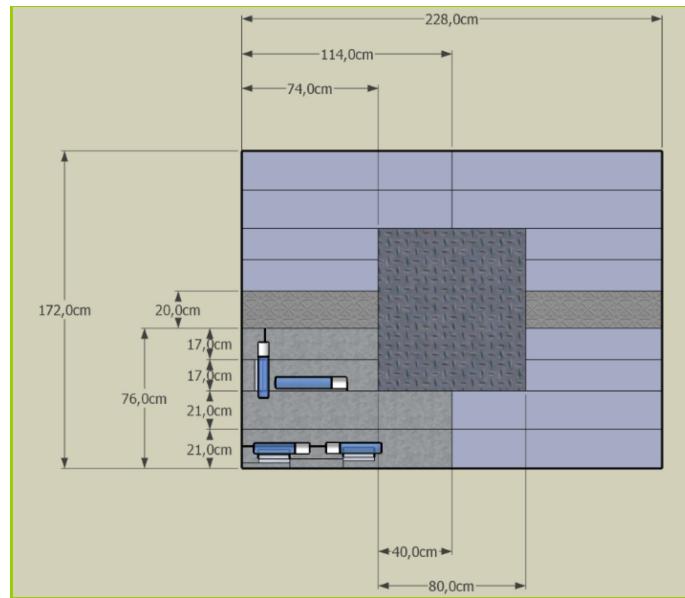


FIGURE 3.28 – Plan du plafond de la chambre

Chaque quadrant possède quatre lampes. Trois de 400 watts 3.29a 3.29b et une de 1000w 3.29c. Ces lampes sont disposées à proximité de réflecteurs, permettant ainsi de pouvoir réfléchir la lumière plus uniformément dans la chambre.

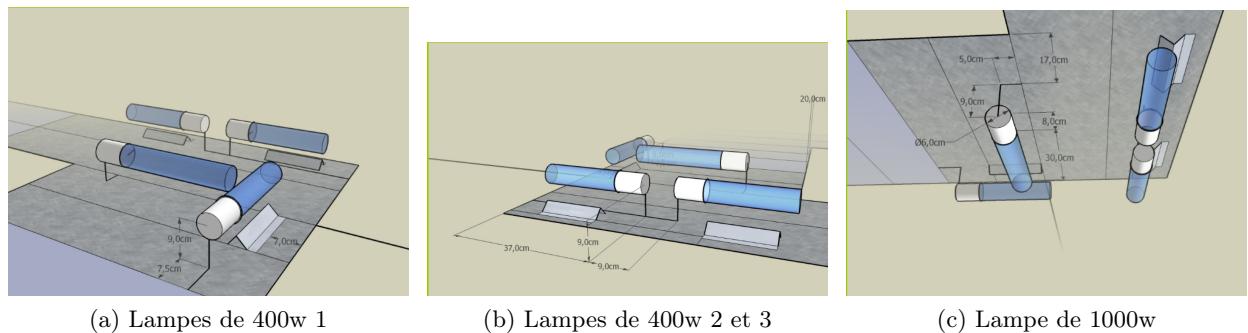


FIGURE 3.29 – Disposition des différentes lampes

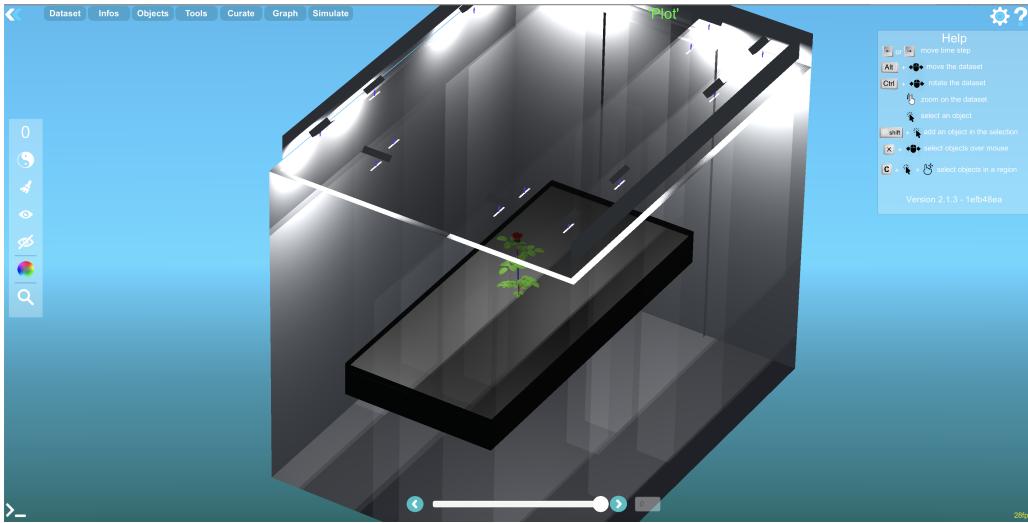


FIGURE 3.30 – Chambre de culture dans MorphoNet

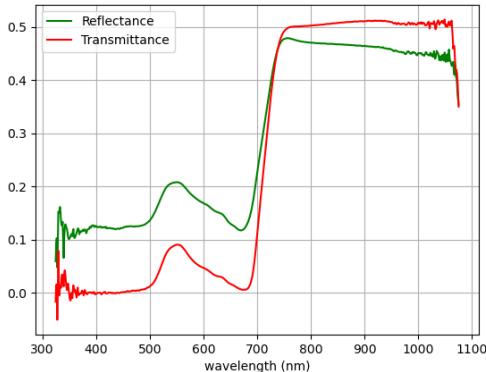
3.3.2 Matériaux de la chambre

Une fois la géométrie mise en place, il a fallu assigner des propriétés optiques à chaque objet.

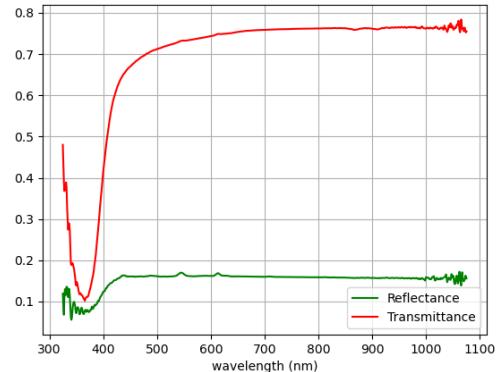
Les matériaux permettent de définir les propriétés physiques des objets 3D. Ils contrôlent la manière dont un objet 3D apparaît à l'écran, à savoir sa couleur, son aspect terne ou sa brillance. On peut également considérer qu'un matériau est un ensemble d'instructions données à un programme de modélisation 3D qui décrit l'apparence du modèle 3D. Par exemple, ils définissent s'il est assez brillant pour faire apparaître un reflet, ou s'il est transparent, et de quelle couleur il est.

Dans notre cas, pour pouvoir simuler le comportement de la lumière selon différentes longueurs d'ondes, on se base sur des spectres de réflectance et de transmittance des matériaux. En effet, on peut voir sur la figure 3.31 que ces valeurs sont déterminées par longueur d'onde.

La réflectance d'un matériau permet de déterminer la quantité de lumière réfléchie par ce dernier. La transmittance permet de déterminer la quantité de lumière transmise par le matériau, un abus de langage commun est d'appeler cela transparence.



(a) Transmittance / réflectance de la face inférieure de feuilles de rose vieille



(b) Transmittance / réflectance du makrolon

FIGURE 3.31 – Réflectance / transmittance de différents matériaux

L’union de ces deux caractéristiques permet de déterminer l’absorbance. Si la somme de la réflectance et de la transmittance ne vaut pas 1, alors 1 moins cette somme est l’absorbance du matériau.

Il a de nombreux confusions sur la différence entre réfraction et transmission. Je vais alors redéfinir ces termes. La réfraction se produit lorsque la trajectoire de la lumière s’infléchit à une frontière lorsqu’elle est transmise dans un nouveau milieu. La transmission se produit lorsque la lumière traverse un milieu, c’est-à-dire à quel point elle est atténue par ce milieu.

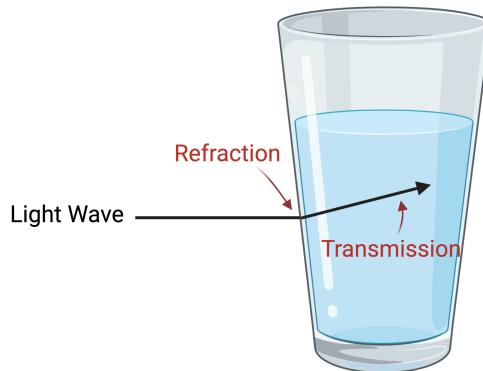


FIGURE 3.32 – Visualisation des phénomènes de réfraction et de transmission

Ces matériaux permettent de déduire une probabilité de comportement d’un rayon lorsqu’il touche une surface. Il y a quatre possibilités de comportement :

- Réflexion spéculaire
- Réflexion diffuse

CHAPITRE 3. TRAVAUX EFFECTUÉS

- Réfraction spéculaire
- Réfraction diffuse

Lorsqu'il y a réfraction ou réflexion spéculaire, l'angle θ entre le rayon incident et sortant sont égaux 3.33, le rayon sortant est unique. Idéalement, l'énergie du rayon incident se retrouve totalement dans le rayon sortant, en pratique une partie de l'énergie peut être absorbée, diffusée ou réfractée au niveau de l'interface.

Lors d'un comportement diffus, le rayon sortant est rediffusé dans un grand nombre de directions 3.34.

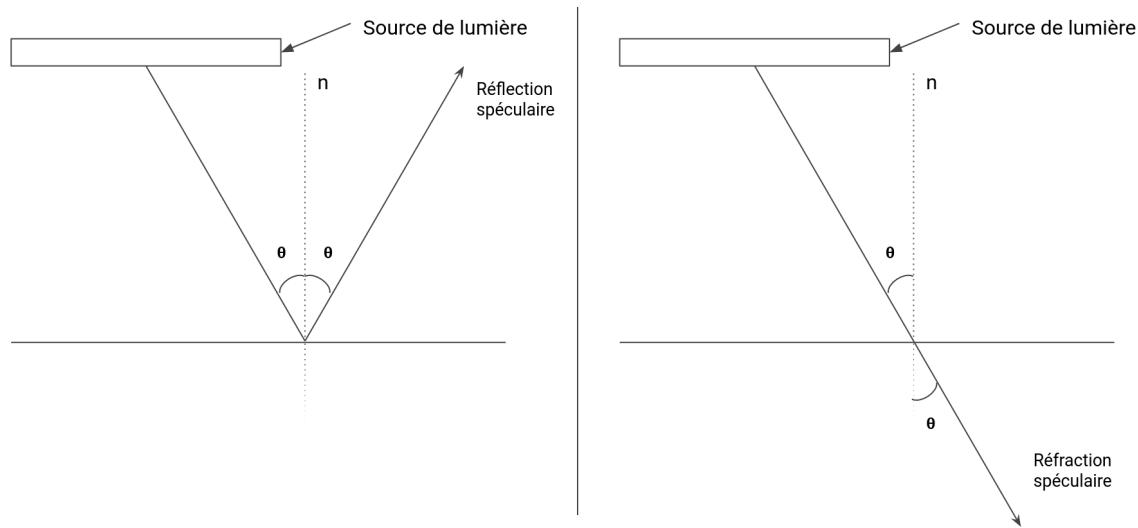


FIGURE 3.33 – Réfraction / réflexion spéculaire

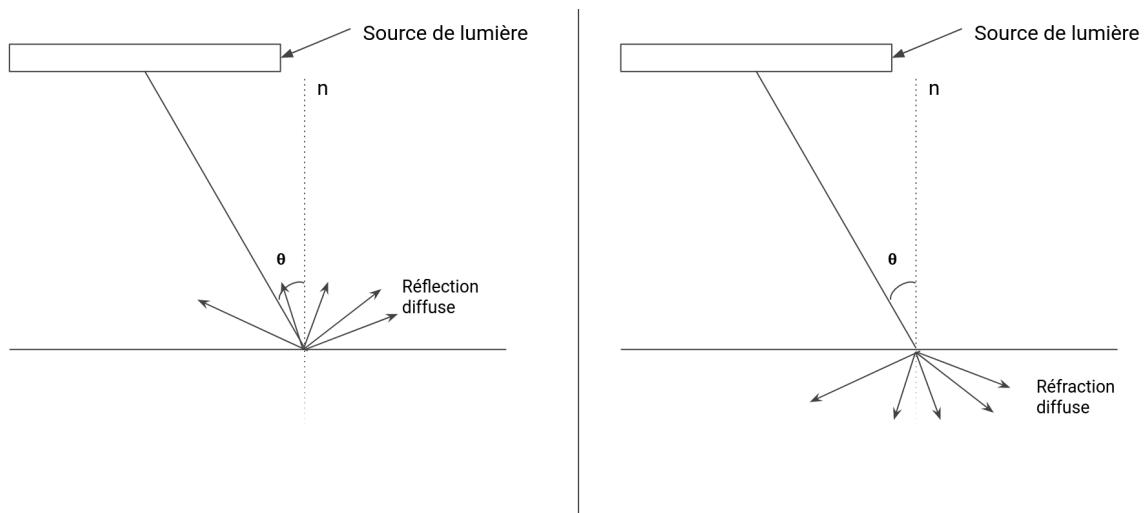


FIGURE 3.34 – Réfraction / réflexion diffuse

Dans notre cas, pour ce qui est des réfractions, nous n'avons que des réfractions diffuses.

Bidirectional scattering distribution function

Pour déterminer la direction du rayon réfléchi / réfracté, on utilise des "Bidirectional scattering distribution function" (Fonction de distribution de la diffusion bidirectionnelle).

Le plus souvent, ce terme est utilisé pour désigner la fonction mathématique générale qui décrit la manière dont la lumière se comporte lorsqu'elle touche une surface. Toutefois, dans la pratique, ce phénomène est généralement divisé en composantes réfléchie et transmise, qui sont alors traitées séparément en tant que BRDF (fonction de distribution de la réflectance bidirectionnelle) et BTDF (fonction de distribution de la transmittance bidirectionnelle). Le fonctionnement de celles-ci est le suivant 3.35 :

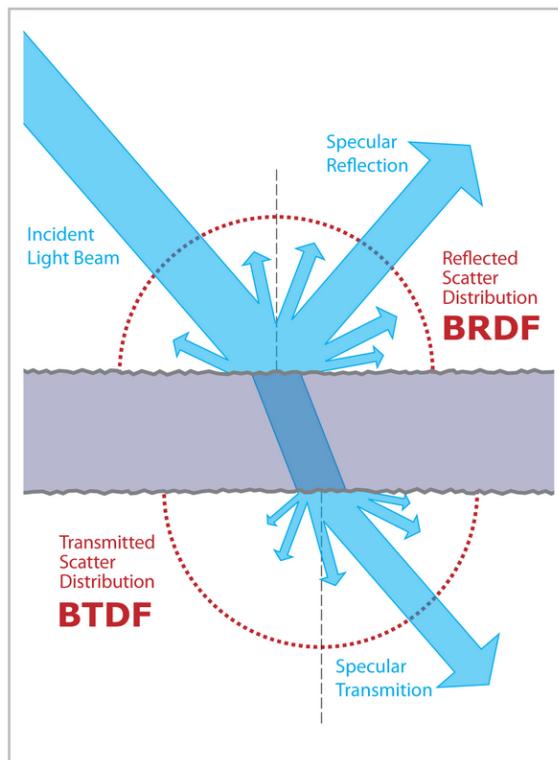


FIGURE 3.35 – Exemple du fonctionnement d'une BSDF¹⁵

Ces fonctions reposent sur le calcul d'une probabilité de transmission ou de réflexion. Pour obtenir cette probabilité, on peut soit utiliser les informations de réflectance et de transmittance des matériaux fournis par les biologistes 3.31, qui est la méthode retenue, soit les calculer à partir de l'indice de réfraction des milieux d'évolution de la lumière.

¹⁵. https://en.wikipedia.org/wiki/Bidirectional_scattering_distribution_function

L'utilisation de l'indice de réfraction permettrait de simuler l'évolution de la lumière au sein même de la plante. Cependant, ne disposant pas des indices de réfraction de la rose ni de modèles 3D volumiques, cette méthode n'est pas celle utilisée bien qu'implémentée.

Pour calculer cette probabilité à partir de l'indice de réfraction, on peut calculer le coefficient de Fresnel pour savoir si la lumière est réfléchie ou transmise.

Le coefficient de Fresnel peut être approché avec l'approximation de Schlick[5] Cette approximation est la suivante :

$$fr(\theta) = f_0 + (1 - R_0)(1 - \cos(\theta))^5 \quad (3.8)$$

Les bornes de ce coefficient sont [0,1]. Une fois ce coefficient obtenu, on utilise l'algorithme suivant 3 :

Algorithme 3 : refract

Début

```

fr ← fresnel(iorI, iorT);
a ← random(0, 1) //Tirer un nombre aléatoire entre 0 et 1.
si fr > a alors
| reflection();
finsi
sinon
| //Test de possibilité de réfraction (assez d'énergie pour changer de milieu).
| th = -iorI/iorT × (v - (dot(v, n) × n));
| si dot(th, th) > 1 alors
| | // Réfraction impossible.
| | reflection();
| finsi
| sinon
| | tp = -(√max(1 - dot(th, th), 0)) × n;
| | direction = th + tp;
| finsi
finsi
Fin

```

3.3.3 Voisinage des plantes

Dans une chambre de culture, les plantes évoluent rarement seules, ce qui impacte la distribution de la lumière. C'est pourquoi il a fallu aussi modéliser le voisinage de la plante étudiée. MorphoNet ne pouvant représenter qu'un seul jeu de données, il faut prendre en compte les voisins que lors de la création de la scène de photon mapping. Pour que l'utilisateur puisse contrôler ce voisinage, il faut une interface. La première interface développée permettait de définir le nombre de voisins dans les directions des quatre points cardinaux. Ce nombre est défini par un slider et permet de construire ce genre de scène 3.36 :

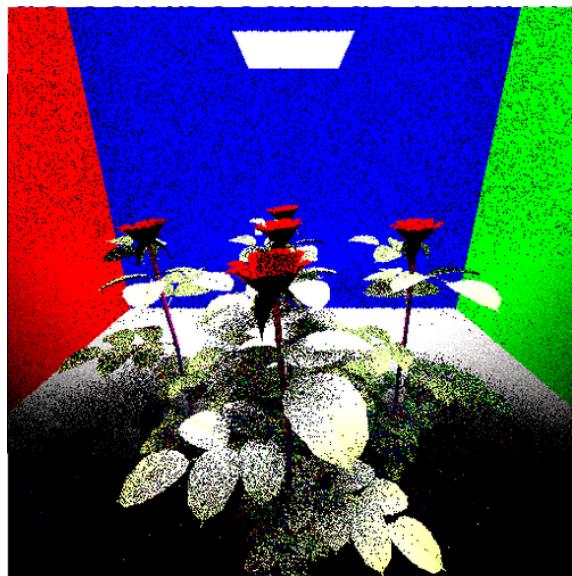


FIGURE 3.36 – Voisinage dans les quatre points cardinaux

Cependant, cette interface n'offrait pas assez de contrôle selon les utilisateurs. Une autre interface doit être développée, elle ne l'est pas encore à l'heure de la rédaction. Elle consistera en un nombre de lignes et de colonnes ainsi que la distance d'espacement.

3.3.4 Capteurs virtuels de photons

Afin de pouvoir valider la simulation de lumière, on souhaite la calibrer avec des données issues de capteurs. Ces capteurs sont ne doivent pas influencer la simulation, on considère qu'ils ont une transmittance égale à 1. Il se présente sous la forme de disque avec comme information une position x, y, z dans la scène 3D, un rayon en centimètres et un vecteur normal pour pouvoir définir leur orientation. Ces capteurs sont définis dans un fichier CSV dont voici un extrait :

```
X;Y;Z;rayon_capteur;Xnorm;Ynorm;Znorm
110;930;1000;10;0;0;1
```

Ces capteurs ne sont pas encore implémentés à l'heure de la rédaction par manque de temps.

3.4 Résultats

Voici le résultat d'une simulation avec 10 000 photons en entrée sur une scène avec 16 lampes, chaque lampe est constituée de quatre triangles émissifs, soit 640 000 photons lancés par pas de temps. La simulation est faite sur la bande verte avec le spectre de la figure 3.25.

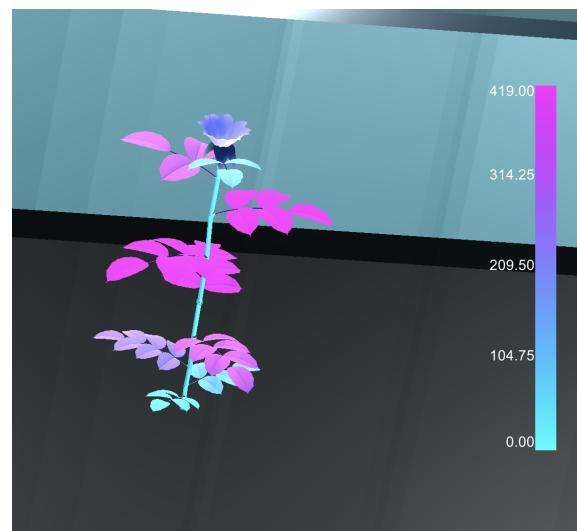


FIGURE 3.37 – Gradient de couleur de l'information d'illumination au pas de temps 150

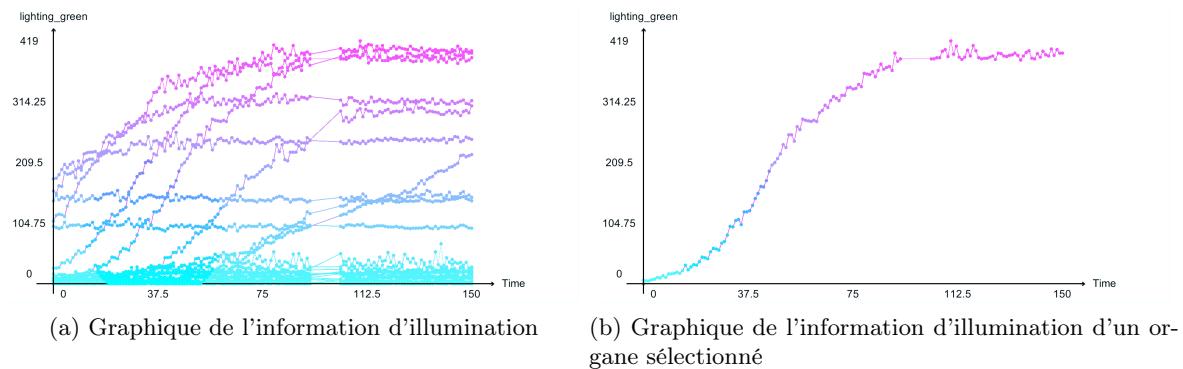


FIGURE 3.38 – Comparaison de l'information de l'illumination globale et de l'illumination d'un organe sélectionné

Comme on peut le voir sur le graphique 3.38a, il y a encore quelques soucis sur la création de l'information d'illumination. Certains pas de temps ont été sautés.

Les résultats précédents ne prennent pas encore en compte toutes les corrections du nombre de photons sur chaque organe défini dans la partie 3.2.4, je n'applique pas encore la correction en fonction de l'intensité. Avant la définition de cette correction, la simulation se faisait avec toutes les lumières en même temps, or comme définie dans cette partie, il faut faire la simulation sur chaque lumière de manière individuelle. J'ai commencé à implémenter cette solution. Je peux obtenir par exemple les photon maps de chaque lampe.

Chapitre 4

Conclusion

La plupart des objectifs du stage ont été atteints. Nous pouvons interagir de manière plus fluide qu'avant avec les outils de simulation et explorer celle-ci. Nous arrivons aussi à reproduire l'environnement et les conditions d'illumination des plantes.

Il reste à tester le module de photon mapping avec une simulation réactive et ajouter les capteurs virtuels pour pouvoir affiner les paramètres du modèle de lumière. Il y a encore quelques soucis sur la création de l'environnement, il faut encore mieux définir comment un environnement est structuré avec nos utilisateurs.

Les perspectives de poursuite de stage seraient d'avoir des maillages volumiques pour pouvoir simuler l'évolution des photons au sein même de la plante. Aussi, j'ai discuté avec certains utilisateurs qui voudraient pouvoir charger un objet et une scène 3D pour calculer l'illumination sans que ce soit une simulation. Cependant, l'objet doit être segmenté pour pouvoir distinguer chacune de ses parties, sinon on ne calculerait que le nombre de photons sur l'entièreté de l'objet. Pour la modélisation des lumières, on pourrait utiliser les fichiers standardisés LDT¹. Ces fichiers permettent de modéliser la distribution de la lumière en fonction d'informations de fournisseurs de lampes 4.1.

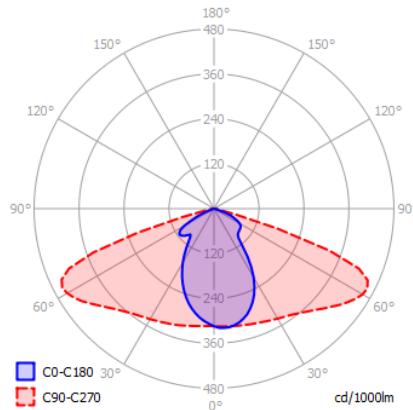


FIGURE 4.1 – Distribution de lumière d'un fichier LDT

En somme, ce stage m'a permis de découvrir le travail en équipe. Il m'a permis aussi d'apprendre à m'insérer dans un projet informatique qui est développé depuis plusieurs

1. <https://www.dialux.com>

CHAPITRE 4. CONCLUSION

années. J'ai aussi appris à communiquer avec des professionnels d'autre milieu que le mien pour travailler ensemble vers un objectif commun.

Ces expériences professionnelles m'ont donné envie de poursuivre dans le milieu de la recherche.

Annexe A

Glossaire

- **LIRMM** : Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier.
- **Cirad** : Centre de coopération internationale en recherche agronomique pour le développement.
- **Ray tracing** : Terme générique qui recouvre les techniques de rendu qui trace des rayons entre une scène et la caméra.
- **Photon Mapping** : En imagerie numérique, le photon mapping ou placage de photons est un algorithme d'illumination globale fondé sur le lancer de rayon (ray tracing) utilisé pour simuler l'interaction de la lumière avec différents objets de manière réaliste.
- **Path tracing** : Algorithme visant à résoudre l'équation de rendu :
$$I(x, x') = g(x, x')[\epsilon(x, x') + \int_s \rho(x, x', x'') \Gamma(x', x'') dx'']$$
- **Caustique** : désigne en optique et en mathématiques l'enveloppe des rayons lumineux subissant une réflexion ou une réfraction sur une surface ou une courbe.
- **IOR (Index of Refraction)** : L'indice de réfraction est une grandeur sans dimension, caractéristique d'un milieu, décrivant le comportement de la lumière dans celui-ci. Le vide a pour indice 1.
- **BTDF** : Bidirectional Transmittance Distribution Function. Fonction permettant de calculer la lumière transmise à travers un milieu.
- **BRDF** : Bidirectional Reflectance Distribution Function. Fonction permettant de calculer la lumière réfléchit par une surface.
- **BSDF** : Bidirectional scattering distribution function. Fonction de distribution de la diffusion bidirectionnelle.
- **Wrapper** : Programme permettant de faire communiquer deux autres programmes en différents langages.
- **PAR (Photosynthetically active radiation)** : désigne la gamme spectrale (bande d'ondes) du rayonnement solaire de 400 à 700 nanomètres que les organismes photosynthétiques sont capables d'utiliser dans le processus de photosynthèse.

Annexe B

Annexes

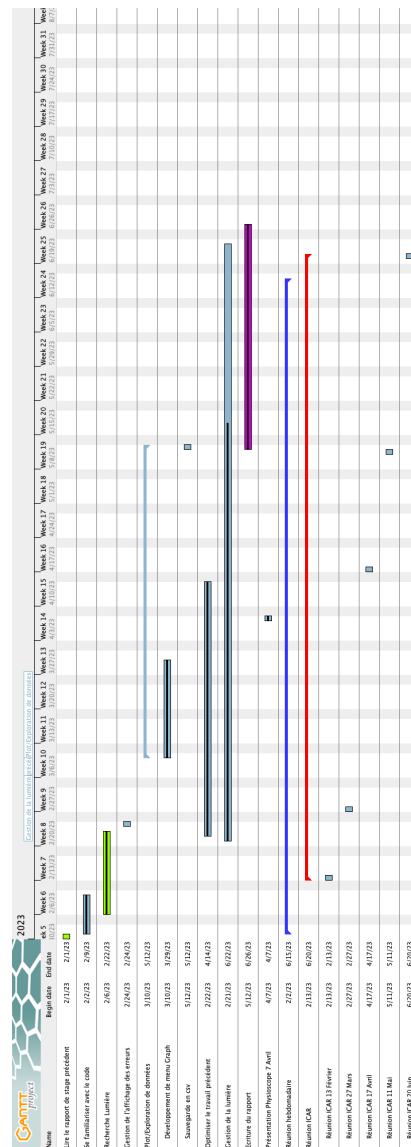


FIGURE B.1 – Diagramme de Gantt

ANNEXE B. ANNEXES

Bibliographie

- [1] A. APPEL, « Some techniques for shading machine renderings of solids, » in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 1968, p. 37-45.
- [2] T. WHITTED, « An Improved Illumination Model for Shaded Display, » *Commun. ACM*, t. 23, n° 6, p. 343-349, juill. 1980, ISSN : 0001-0782. DOI : 10.1145/358876.358882. adresse : <https://doi.org/10.1145/358876.358882>.
- [3] J. T. KAJIYA, « The Rendering Equation, » *SIGGRAPH Comput. Graph.*, t. 20, n° 4, p. 143-150, août 1986, ISSN : 0097-8930. DOI : 10.1145/15886.15902. adresse : <https://doi.org/10.1145/15886.15902>.
- [4] S. CRONE, « Radiance users manual, » *Architectural Dissertation*, t. 2, 1992.
- [5] C. SCHLICK, « An Inexpensive BRDF Model for Physically-based Rendering, » *Computer Graphics Forum*, t. 13, n° 3, p. 233-246, 1994. DOI : <https://doi.org/10.1111/1467-8659.1330233>. eprint : <https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.1330233>. adresse : <https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.1330233>.
- [6] G. J. WARD, « The RADIANCE lighting simulation and rendering system, » in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, p. 459-472.
- [7] H. W. JENSEN, « Global illumination using photon maps, » in *Rendering Techniques' 96 : Proceedings of the Eurographics Workshop in Porto, Portugal, June 17–19, 1996* 7, Springer, 1996, p. 21-30.
- [8] M. CHELLE, C. RENAUD, S. DELEPOULLE et D. COMBES, « Modeling light phyllotaxis within growth chambers, » in *5. International Workshop*, sér. Functional-Structural Plant Models. Abstracts of papers and posters, Napier, New Zealand, nov. 2007, np. adresse : <https://hal.science/hal-01191959>.
- [9] C. PRADAL, F. BOUDON, C. NOUGUIER, J. CHOPARD et C. GODIN, « PlantGL : A Python-based geometric library for 3D plant modelling at different scales, » *Graphical Models*, t. 71, n° 1, p. 1-21, 2009, ISSN : 1524-0703. DOI : <https://doi.org/10.1016/j.gmod.2008.10.001>. adresse : <https://www.sciencedirect.com/science/article/pii/S1524070308000143>.
- [10] F. BOUDON, C. PRADAL, T. COKELAER, P. PRUSINKIEWICZ et C. GODIN, « L-Py : An L-System Simulation Framework for Modeling Plant Architecture Development Based on a Dynamic Language, » *Frontiers in plant science*, t. 3, p. 76, mai 2012. DOI : 10.3389/fpls.2012.00076.

BIBLIOGRAPHIE

- [11] M.-G. RETZLAFF, J. HANIKA, J. BEYERER et C. DACHSBACHER, « Physically based computer graphics for realistic image formation to simulate optical measurement systems, » *Journal of Sensors and Sensor Systems*, t. 6, p. 171-184, mai 2017. DOI : [10.5194/jsss-6-171-2017](https://doi.org/10.5194/jsss-6-171-2017).
- [12] B. LEGGIO, J. LAUSSU, A. CARLIER, C. GODIN, P. LEMAIRE et E. FAURE, « MorphoNet : an interactive online morphological browser to explore complex multi-scale data, » *Nature Communications*, t. 10, p. 2812, juin 2019. DOI : [10.1038/s41467-019-10668-1](https://doi.org/10.1038/s41467-019-10668-1).