Faculté des Sciences

Master 2 Informatique
IMAGINA

Rapport de stage

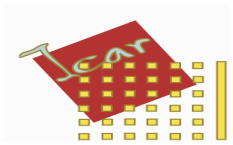
Reconstruction de l'architecture d'un arbre à partir de photographies numériques

effectué au CIRAD du 12 février au 11 août 2017 par
Chouenyib Ali

Encadrant de stage :
M. Philippe Borianne

co-encadrant :
M. Gérard Subsol

Enseignant référent :
M. Konstantin Todorov



27 juin 2017

Remerciements

Mes remerciements vont, tout d'abord à M. Philippe Borianne, mon maître de stage, pour m'avoir permis d'effectuer ce stage dans un environnement stable, pour sa disponibilité, pour son encadrement inconditionnel et efficace, et pour les précieuses recommandations et indications qu'il m'a donné durant ce stage.

Je remercie également mon professeur d'université M. Gérard Subsol qui m'a permis d'avancer dans ma recherche.

Enfin, j'adresse mes remerciements à toutes les personnes qui m'ont fait partager leur expérience et/ou accorder leur soutien au cours de ce stage.

Résumé

Le diagnostic in situ de l'état de développement de l'arbre urbain passe entre autres par le dénombrement et la spatialisation d'éléments clé de la lecture architecturale de la structure arborescente, notamment les bourgeons, les gourmands, les rejets et les branches mortes. Le but de ce stage est de développer une application permettant de reconstruire l'architecture d'un arbre à partir de photographies numériques. Dans ce rapport, nous présentons la travail effectué dans ce sens pendant mon stage au CIRAD.

Mots clés : images, arbre botanique, squelettisation, graphe, arbre mathématique, architecture.

Sommaire

Table de Figures

Acronymes

BL Berger Levrault, page 6

ACP Analyse en Composantes Principales, page 15

AMAP botAnique et Modélisation de l'Architecture des Plantes et des végétations, page 8

Bios Systèmes Biologiques, page 7

CIRAD Centre de coopération Internationale en Recherche Agronomique pour le Développement, page 6

CNRS Centre National de la Recherche Scientifique, page 8

ES Environnement et Sociétés, page 7

INRA Institut National de la Recherche Agronomique, page 8

IRD Institut de Recherche pour le Développement, page 8

LIDAR Light Detection And Ranging, page 11

Persyst Performances des Systèmes et Transformation tropicaux, page 7

RVB Rouge Vert Bleu, page 14

UMR Unité Mixte de Recherche, page 6

UM Université de Montpellier, page 8

Introduction

Dans les zones urbaines, les arbres sont plantés, essentiellement pour des raisons écologiques ou esthétiques. Cependant un arbre requiert un suivi et des traitements au cours de sa croissance. Estimer l'état sanitaire d'un arbre urbain s'appuie sur une méthodologie de diagnostic basée sur la connaissance de l'architecture de la structure arborescente, notamment sur l'organisation de la ramification de la couronne et la distribution spatialisée des bourgeons, des gourmands, des rejets et des branches mortes. C'est sur cette dernière que nous focaliserons notre attention.

En effet, le diagnostic vise à maintenir la vitalité des arbres et à prévenir les anomalies au cours de leur développement. Au CIRAD, des chercheurs travaillent sur des solutions automatiques pour permettre aux botanistes de réaliser rapidement en terme de temps un diagnostic.

Ce stage est effectué au sein de l'UMR AMAP. Il s'inscrit dans le cadre du lancement d'un projet exploratoire **BL/CIRAD**. Mon travail se base sur l'étude, l'implémentation et la validation d'un processus algorithmique permettant de faire un diagnostic architectural d'un arbre urbain à partir de photographies numériques.

Ce rapport présente la synthèse du travail effectué durant mon stage. Nous présentons tout d'abord l'entreprise d'accueil et les objectifs du stage, nous définissons en détails le problème posé, puis une étude de l'existant pour comprendre les principes et justifier le choix de notre approche. Ensuite nous faisons une synthèse de la solution mise en place et implémentée. Enfin nous présentons les résultats obtenus, qui feront l'objet d'une discussion et de perspectives.

1. Présentation de l'entreprise

1.1 Structure et organisation de l'entreprise

1.1.1 Le CIRAD

Le CIRAD est un établissement public à caractère industriel et commercial. Il a été créé en 1984. Le CIRAD est placé sous la double tutelle du ministère de l'enseignement supérieur et du ministère des affaires étrangères et européennes.

Mission

La mission du CIRAD est de produire et de transmettre de nouvelles connaissances pour accompagner l'innovation et le développement agricole. Il a comme objectif prioritaire de bâtir une agriculture durable, adaptée aux changements climatiques tout en préservant l'environnement.
Le CIRAD est aussi un acteur majeur du dialogue entre l'Europe et l'Afrique. Il facilite l'accès à ces partenaires du sud aux programmes communautaires et leur insertion dans les réseaux internationaux de coopérations scientifiques.

Le CIRAD en chiffres

Le CIRAD emploie 1650 personnes, dont 800 chercheurs. Il comprend 3 départements scientifiques, regroupant chacun une dizaine d'unités (propres ou mixtes) de recherche :

- le département scientifique Systèmes Biologiques (**Bios**)
- le département scientifique Performances des Systèmes de production et de transformation tropicaux (**Persyst**)
- le département scientifique Environnements et Sociétés (**ES**).

Le CIRAD est constitué par 33 unités de recherche.

1.1.2 l'UMR AMAP

L'UMR AMAP associe plusieurs disciplines, en particulier la Botanique, l'Écologie, les Mathématiques Appliquées et l'Informatique Scientifique. Cette association est un enjeu scientifique et technique majeur pour l'organisme et la foresterie moderne. l'UMR AMAP est sous la tutelle de 5 organismes :

- Le Centre de Coopération Internationale de Recherche Agronomique pour le Développement (CIRAD)
- le Centre national de la recherche scientifique (CNRS)
- l'Institut National de la Recherche Agronomique (INRA)
- l'Institut de Recherche pour le Développement (IRD)
- l'Université de Montpellier (UM)

Elle développe et évalue des modèles d'analyse et de suivi des paramètres morphologiques, anatomiques et fonctionnels du développement architectural des espèces végétales. Elle crée des logiciels, dont certains sont commercialisés, et diffuse ses résultats auprès des chercheurs et étudiants, qu'elle accueille et forme.

1.2 Objectif du stage

L'objectif de ce stage est de développer une application permettant de reconstruire l'architecture d'un arbre extrait à partir de photographies numériques 2D (Exemple : fig 1.1).

Dans ce stage, nous nous focalisons sur la reconstruction de l'architecture de l'arbre extrait dans une image numérique 2D.

L'étude bibliographique permettra de comprendre les principes utilisés dans ces différentes problématiques et d'identifier les avantages et inconvénients des solutions proposées dans la littérature. Nous implémenterons par la suite en Java sur ***ImageJ***[1] la solution la plus adaptée. En fin, un jeu de données représentatives constitué de

[1]ImageJ : Un logiciel multiplate-forme et open source de traitement et d'analyse d'images. (`https://imagej.nih.gov/ij/docs/index.html`)

FIGURE 1.1 – Image numérique d'un jeune charme - Montpellier - février 2017
légende : A cette saison, l'arbre ne présente pas de feuille, ce qui est plus adapté pour l'étude de la ramification. Malgré son jeune âge, environ 20 ans, notez la complexité de arborescence et les variations de couleurs et de saturation des branches.

photographies prises in situ sera produit pour apprécier les résultats de la méthode implémentée.

2. Le Problème

2.1 Construction de l'architecture d'un arbre

Le stage est centré sur l'aspect architectural, plus exactement sur son utilisation dans l'évaluation de l'état de développement de l'arbre à partir des données numériques. L'architecture contribue à apprécier la dynamique de la structure arborescente de l'arbre dans le temps et dans l'espace (fig 2.1), et de fait à prédire le devenir de l'arbre.

L'architecture revêt implicitement deux aspects : la reconstruction de la structure arborescente de l'arbre à partir d'images numériques, et le dénombrement et l'analyse de la distribution des axes caractérisant la structure.

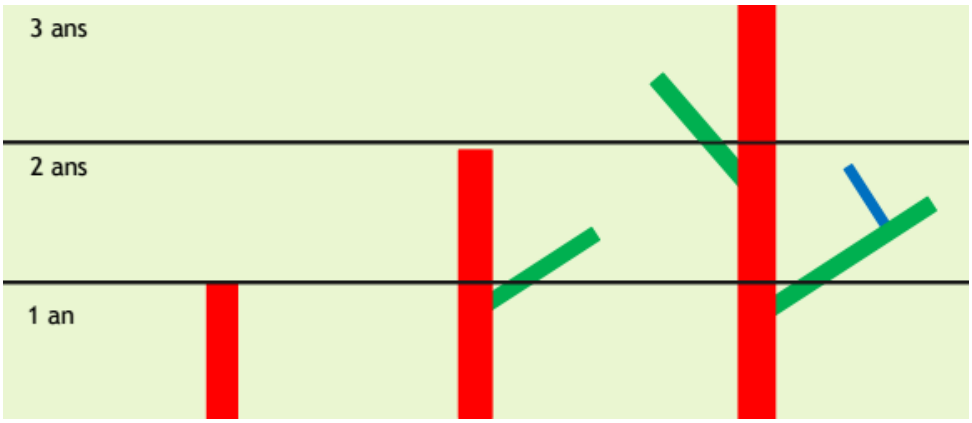


FIGURE 2.1 – Architecture d'un arbre : ordre de ramifications dans le temps

Dans la figure 2.1, en rouge le tronc (d'*ordre de ramification 0*), en vert les axes d'*ordre 1* portés par le tronc et en bleu les axes d'*ordre 2* portés par les axes d'ordre 1.

Année 1 : mise en place du tronc ;

année 2 : le tronc mis en place l'année précédente s'allonge du fait du développement du bourgeon apical ; parallèlement à cet allongement, une ramification apparaît du

fait du développement du bourgeon latéral et donne naissance à un axe d'*ordre 1*.
Année 3 : les axes mis en place l'année précédente s'allongent et se ramifient : un axe d'*ordre 2* apparaît sur la partie la plus ancienne de l'axe d'*ordre 1*, l'architecture de l'arbre se met en place.

2.2 Étude de l'existant

La reconstruction de l'architecture d'un arbre peut être basée, soit sur des règles ou une grammaire pour la création des modèles, (par exemple les L-System, grammaire formelle adaptée pour la modélisation du processus de développement des plantes [1]), soit avec des images, où le processus de modélisation s'effectue directement en utilisant un échantillon d'images [2].

Il existe des méthodes permettant de créer des modèles réalistes pour les arbres. Imiter la forme d'un arbre requiert beaucoup de paramètres et parfois même, des ajustements manuels. Ces méthodes se distinguent, par plusieurs informations. certaines travaillent avec plusieurs images prises sous différents angles, pour le même arbre, d'autres avec des acquisitions **LIDAR**, ou bien à partir d'une seule image couleur 2D comme [2].

[3] utilisent dans leur méthode plusieurs images pour reconstruire l'architecture d'un arbre. L'idée principale est de combiner une construction ascendante avec des contraintes externes et internes. La méthode proposée peut être vue comme une simulation de particules avec des contraintes externes à partir d'images et des restrictions botaniques internes. Les résultats obtenus reflètent bien la structure et les branchements. En étant capable d'adapter la simulation de particules à un ensemble d'images, cela a permis de créer des modèles d'arbres assez proches de la réalité.

D'autres approches de modélisations exploitent l'intensité, les nuages de points et/ou l'information de profondeur à partir des données LIDAR. Ces dernières requièrent par contre plusieurs paramétrages et ne récupèrent pas de manière réaliste les zones incomplètes. [4] ont introduit une méthode de reconstruction d'arbre à partir des nuages de points. La fonctionnalité principale de la méthode est la capacité à générer de manière automatique les branches des zones incomplètes sans aucune segmentation.

Il y a également certaines méthodes qui n'utilisent qu'une seule image pour reconstruire l'architecture d'un arbre. C'est le cas de la méthode de [2], cette dernière

a pour objectif la reconstruction d'une large gamme de végétations et la simplification du processus de modélisation. La méthode génère automatiquement un modèle 3D avec les branches et les feuilles. Les branches sont synthétisées par un moteur de croissance à partir d'une librairie de petites sous-branches prédéfinies ou récupérées à partir des branches visibles. Les feuilles sont générées à partir de la région délimitée par le houppier pour compléter l'arbre.

[5] quant à lui propose un processus d'analyse de l'architecture des racines à partir d'images en niveaux de gris. Nous nous sommes intéressés à cette méthode pour la résolution de notre problème car bien qu'elles soient enfouies dans la terre, les racines tout comme les branches se présentent sous forme d'arborescences en 3D. Par ailleurs cette similarité, dans la méthode de [5], les racines sont analysées à partir de photographie 2D.

2.3 Pipeline de traitement

Nous présentons dans cette partie l'enchaînement de notre méthode. Elle s'appuie en partie sur la méthode proposée par [5]. La méthode est divisée en plusieurs parties de traitements différents, selon le type d'information en entrée et le résultat en sortie, jusqu'au résultat final, qui est la reconstruction de l'architecture. Chaque étape utilise une information précise, et fournit un résultat qui va être utilisé comme information pour la partie suivante (fig 2.2).

La division de l'application en plusieurs traitements différents permet de simplifier aussi sa maintenance, afin de pouvoir modifier/améliorer chaque partie avec plus de simplicité.

Les rectangles en bleu-ciel (fig 2.2) sont les traitements. Ils se font pour chaque image d'arbre lu en entrée, et produisent un nouvel résultat (rectangles en orange). La nature des résultats sont différents selon l'étape du traitement. En effet, à partir de la première partie jusqu'à l'entrée de la partie "***passage au graphe***", les résultats en sortie sont des images en niveaux de gris et/ou binaires. Les trois dernières parties ne traitent pas des données de type image, mais des données topologiques, définissant un graphe mathématique.

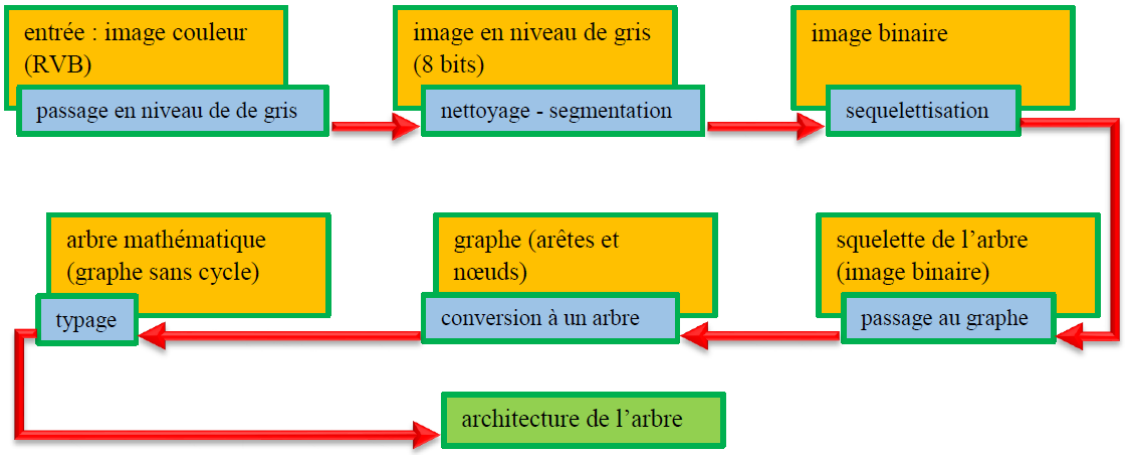


FIGURE 2.2 – Pipeline de traitement

2.4 Méthodologie et Technologies utilisées

2.4.1 Méthodologie

Le travail que nous présentons dans ce document est un travail complexe. C'est un travail de recherche et de développement. Nous n'avions pas un cahier de charge au préalable. En revanche les objectifs étaient fixés, ainsi que les contraintes, notamment technologiques. Pour cela, nous faisons des études bibliographiques, ensuite essayons de mettre en œuvre une méthode et la tester avant de passer à une étape suivante.

2.4.2 Technologies utilisées

Nous avons implémenté l'application en java, sur l'éditeur ***Eclipse***, en utilisant une approche orientée objet. Ce dernier nous a permis de mieux séparer les tâches et avoir un code clair et facile à modifier ou améliorer. L'application a été implémentée sous forme d'un plugin d'***ImageJ***.

3. Synthèse de la solution

Dans cette partie, nous présentons en détail, chaque étape importante du pipeline de traitement de la solution implémentée. Nous avons mis en **annexe B**, les algorithmes (pseudo-codes) des méthodes importantes du pipeline.

3.1 Passage d'une image couleur à une image binaire

L'acquisition des images d'arbres dans des milieux urbains représentent certaines contraintes, surtout quand on souhaite capturer une grande partie de l'arbre (du tronc aux dernières feuilles du houppier), ou bien mieux isoler l'arbre de différents objets présents dans son environnement (des bâtiments, lampadaires, d'autres arbres ...) pour avoir un fond assez homogène.

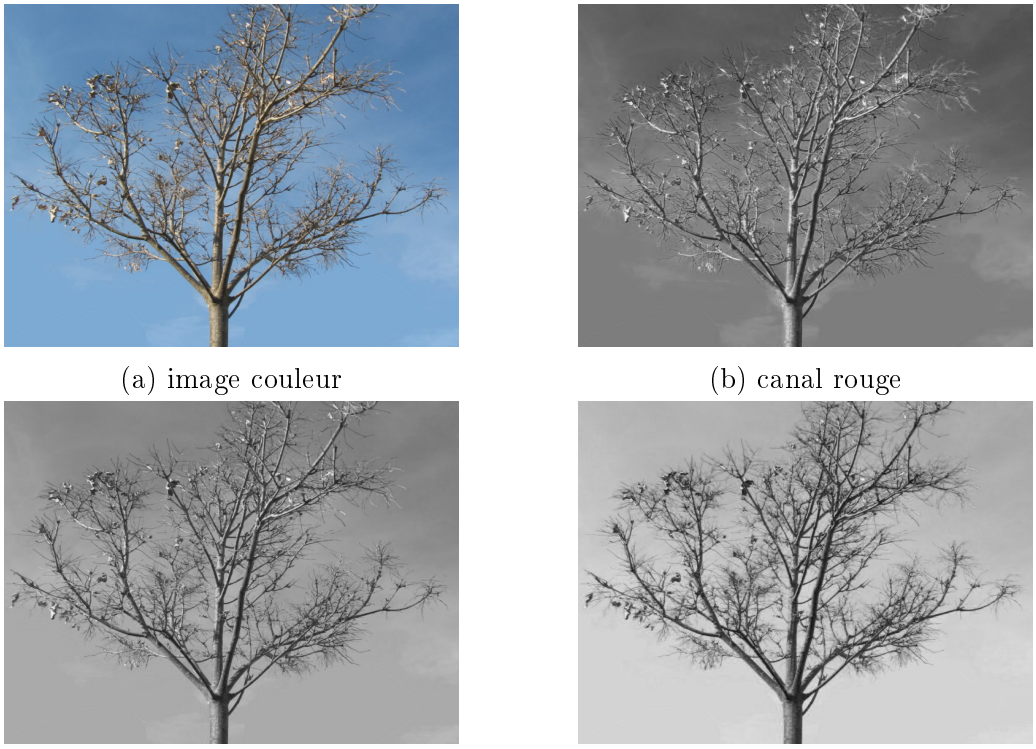
Dans un premier temps, nous avons choisi des images moins complexes. Cela nous permet de ne pas s'attarder dans la phase de segmentation. La segmentation pourrait faire l'objet d'un travail à part. Elle est une partie importante dans notre dans notre pipeline, mais pas un objectif.

3.1.1 Passage en niveaux de gris

Les données d'entrée sont des images couleurs. Nous avons besoin dans l'enchaînement du pipeline de segmenter l'image. Il existe plusieurs méthodes pour passer d'une image couleur à une image en niveau de gris, par exemple en en attribuant à chaque pixel l'intensité moyenne calculée à partir des trois canaux R, V et B, ou faire une **ACP**[1] pour trouver la combinaison linéaire préservant le maximum d'informations et le maximum de contrastes.

[1] **ACP** est une méthode de la famille de l'analyse de données, elle transforme des variables corrélées en nouvelles variables décorrélées les unes des autres.

Ce passage est nécessaire dans notre solution, mais pas un point d'étude en soi, nous avons choisi d'utiliser le canal bleu. En effet, il couvre un intervalle de fréquence allant de 400 à 500 nanomètres pour lequel la fonction d'absorption de la carotène et de la chlorophylle, les principaux composants chimiques du bois et des feuilles, est la plus représentative[2].



(a) image couleur (b) canal rouge

(c) canal vert (d) canal bleu

FIGURE 3.1 – séparation d'une image couleur RVB en 3 canaux

Le canal bleu n'est pas forcément le meilleur, nous avons choisi le canal bleu en suivant les critères cités ci-dessus, sachant que peu importe le choix que nous aurons fait, nous perdrons des informations que nous trouverons dans un autre canal, et inversement.

3.1.2 La segmentation

La segmentation est l'une des étapes importantes et critiques de l'analyse d'images. Elle conditionne la qualité des calculs effectués ultérieurement. Elle permet d'isoler dans l'image les objets sur lesquels doit porter l'analyse, de séparer dans une image

[2] `http://www.snv.jussieu.fr/bmedia/Photosynthese-cours/04-pigments.htm`

les régions d'intérêt du fond. Plusieurs techniques existent, la plus basique et simple étant le seuillage des valeurs de niveau de gris de l'image.

La segmentation peut être effectuée manuellement ou automatiquement. Nous avons opté pour la segmentation automatique, elle permet à l'application de s'adapter aux spécificités des images que nous traitons. Les paramètres de seuillage sont automatiquement adaptés aux contenus et à l'exposition de l'image. Cette adaptabilité de l'algorithme permet d'envisager de traiter des lots d'images. La segmentation automatique s'effectue, soit par seuillage automatique, soit par application d'un filtre passe-haut sur l'image.

Le seuillage automatique est basé sur l'analyse de la distribution de fréquence (cumulée ou histogramme) des niveaux de gris associés à l'image. Il existe plusieurs méthodes pour effectuer un seuillage. Dans notre traitement, nous avons choisi la méthode d'**Otsu**[6]. Elle regroupe les intensités en deux classes en minimisant la variante intra-classe et maximisant la variance inter-classe (exemple : fig 3.2).

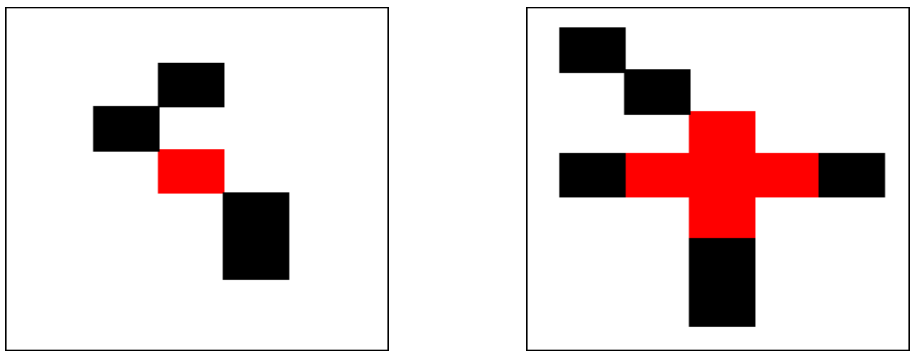
(a) (b)

FIGURE 3.2 – segmentation : (a) image en niveau de gris, (b) image segmentée par seuillage automatique

3.2 Squelettisation

Le processus de squelettisation est une des étapes clés dans le processus de notre pipeline. La squelettisation est une représentation d'une forme en filiforme, en conservant les propriétés topologiques de la forme qu'il représente. Il est généralement défini comme étant l'ensemble des lignes médianes. l'idée consiste à centrer dans la forme un squelette qui soit significatif de l'élongation et des déformations de celle-ci.

Le squelette est une ligne ****-connexe*** de pixels décrivant une arborescence. *-connexe signifie que les lignes sont majoritairement 8-connexe, sauf de manière très localisée où la topologie passe en 4-connexe : la 8-connexité correspond à des enchaînements simples de pixels où chaque pixel a exactement deux voisins : ces enchaînements décrivent des axes curvilignes (fig 3.3a) ; la 4-connexité correspond à des amas correspondant à des branchements d'axes curvilignes (fig 3.3b).



(a) (b)

FIGURE 3.3 – Types de pixels (en rouge) présents dans un squelette : (a) pixel 8-connexe, (b) pixels 4-connexe formant un amas

Il existe plusieurs méthodes de squelettisation, classées en trois familles différentes selon [7] :

- Les algorithmes basés sur le diagramme de ***Voronoi*** ou les approches continues géométriques de nuage de points.
- Les algorithmes basés sur le principe de l'évolution continue de bords des objets, où le squelette est formé aux positions de singularités.
- Les algorithmes basés sur le principe de l'érosion morphologique binaire ou la position des points de singularités sur une carte de distance.

Nous avons choisi la troisième famille, notamment l'érosion puisque la méthode [8] implémentée dans ***ImageJ*** en fait partie. Le principe de cette méthode consiste à effectuer une érosion[3] itérative de la composante connexe(l'objet à squelettiser dans l'image) en suivant des règles topologiques et géométriques.

Malheureusement, le squelette pour être fidèle à la forme initiale, est très sensible au bruit (petite déformation de contour, présence d'un trou) (fig 3.4). Ces bruits peuvent varier d'une méthode (ou implémentation) à une autre. Nous nous sommes interrogés sur les bruits de la méthode que nous utilisons, en l'occurrence la méthode de [8] implémentée dans ***ImageJ***.

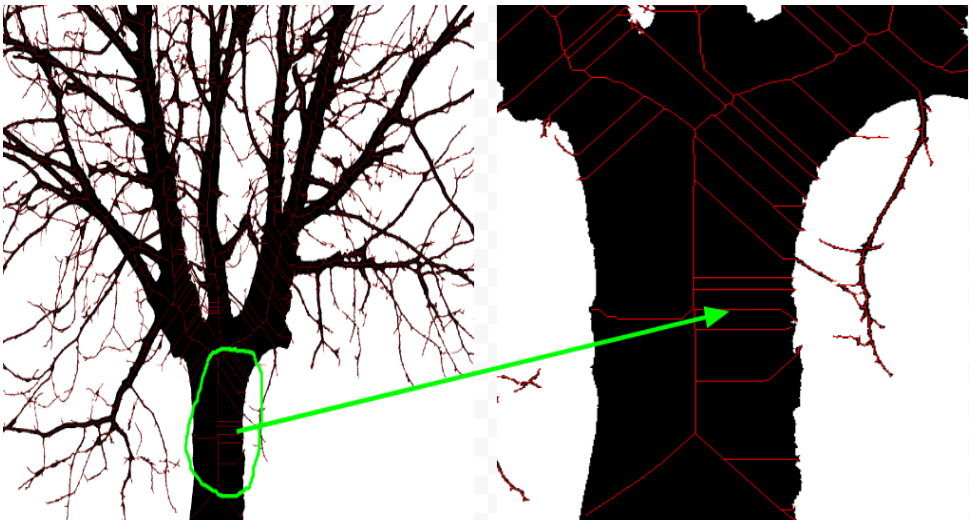


FIGURE 3.4 – lignes présentes en cas de présence de concavités
les concavités locale sur le bord de la région font apparaître des lignes médianes (lignes fuyantes) dont le seul but est de garantir l'intégrité topologique du squelette.

Nous avons essayé d'apporter des éléments de réponse à la question posée ci-dessus (annexe C). Suite à ces expérimentations, la question de fond est de savoir à quel étape du pipeline faudra-t-il nettoyer les artefacts de la squelettisation, en particulier les "lignes ne correspondant pas à des structures botaniques" ?
Nous avons au final reporté ce nettoyage au niveau du graphe. Nous pouvions le faire par exemple sur l'image binaire, en remplissant les petits trous blancs afin de réduire la précision de l'arborescence décrite dans l'image binaire. Mais nous nous sommes rendus compte après quelques expérimentations que nettoyer l'image laisser quand même un biais, et qu'il faudrait encore nettoyer le graphe.

[3]Érosion est l'effet de rétrécir la figure, la figure érodée est la différence de la figure originelle et d'un élément structurant

FIGURE 3.5 – Le squelette de l'arbre, représenté par les traits rouges

Les traits rouges dans la figure 3.5 représente le squelette de l'arbre dans la figure 3.1

Propriétés d'un squelette

les points du squelette porte implicitement des informations de diverses natures (géométrique, colorimétrique ou topologique) obtenues par simple intersection entre l'image du squelette et toute information spatialisée dérivée de l'image native : carte de distance, canaux colorimétriques Rouge, Vert Bleu, Teinte, Saturation, Luminence, etc.... Il peut servir de masque pour la forme de l'objet initial. Il peut en particulier intersecter avec des images. Nous allons énumérer dans cet paragraphe

certaines informations que nous pouvons obtenir à partir du squelette, sans détailler leurs utilités, que nous verrons ultérieurement. Pour chaque pixel du squelette nous avons :

1. ses coordonnées *(x,y)* dans l'image,
2. sa couleur dans l'image couleur (RVB),
3. sa distance au bord le plus proche de l'objet (obtenue par une carte de distance)[9].
4. une typologie définie par le nombre de voisins directs dans le squelette :
   - pixel **extremité** si un seul voisin,
   - pixel d'**arête** si exactement deux voisins,
   - pixel d'**amas** si au moins trois voisins.

3.3 Passage du squelette au graphe

Nous allons ici générer un graphe mathématique à partir du squelette d'une image d'arbre. Nous nous appuyons sur le type du pixel (*-connexe) pour déterminer les éléments caractérisant un graphe.

Un graphe fini $\boldsymbol{G = (N, A)}$ est défini par l'ensemble fini $N = \{n_1, n_2, ..., n_n\}$ dont les éléments sont appelés **nœuds**, et par l'ensemble fini $A = \{a_1, a_2, ... a_m\}$ dont les éléments sont appelés **arêtes**. Une arête a est définie par une paire non ordonnée de nœuds. Ce sont les extrémités de l'arête a.

Un graphe est composé de nœuds reliés deux à deux par des arêtes. Le squelette récupéré à la fin de l'étape précédente du pipeline peut être représenté par un graphe, L'objectif ici est de passer d'une image binaire (le squelette) à une représentation topologique (fig 3.6).

Le passage du squelette au graphe consiste à associer les amas de pixels aux noeuds et les lignes curvilignes aux arêtes d'un graphe.

L'algorithme que nous avons mis en place va donc chercher tout d'abord les éléments (les amas) qui correspondent aux nœuds dans le squelette. Ensuite, il va chercher les éléments qui relient un nœud à un autre. Ces derniers sont les arêtes.

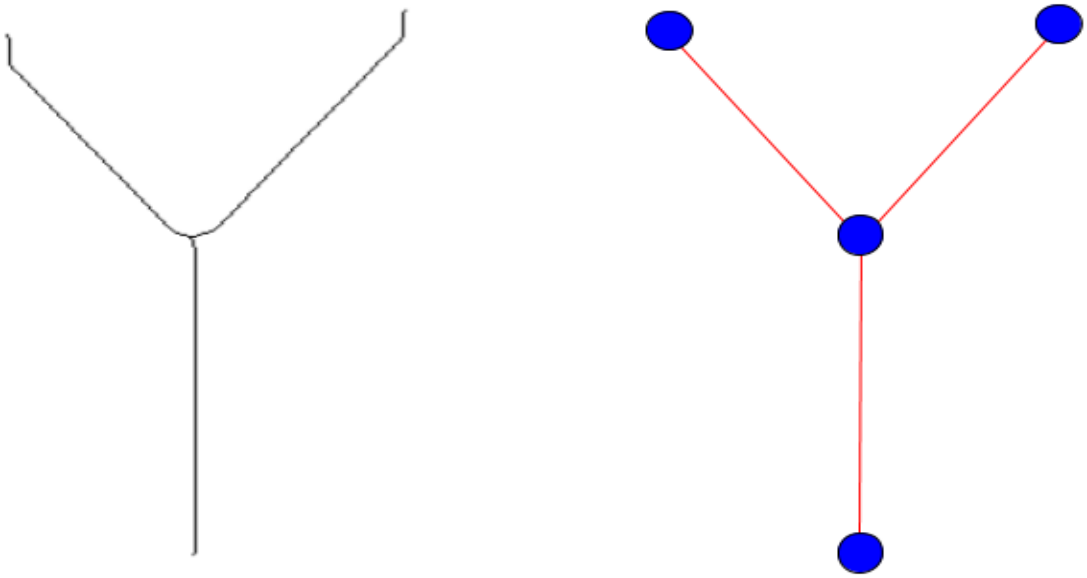


FIGURE 3.6 – Passage du squelette au graphe
légende : à gauche, une image binaire d'un squelette,
à droite le graphe correspondant au squelette.

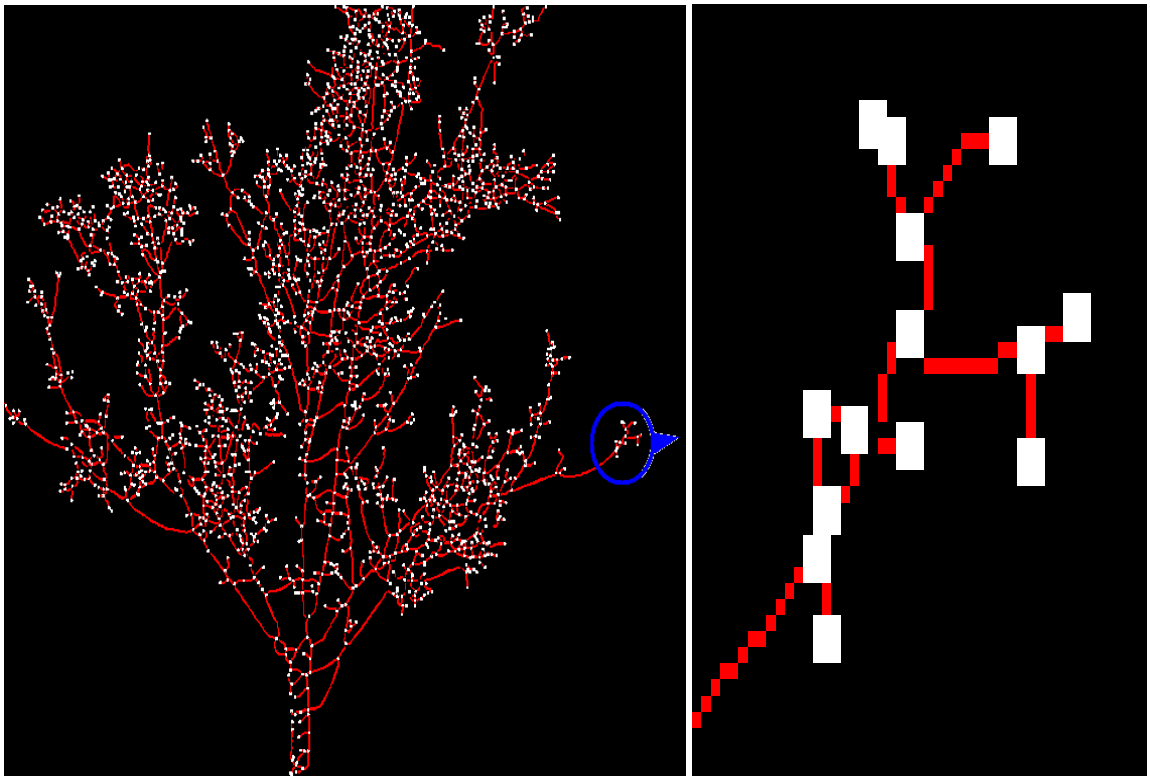
3.3.1 La recherche de nœuds

Pour déterminer les nœuds du graphe, nous cherchons tous les pixels de type **extrémité** et les amas. Un amas, est une composante connexe composé de de pixels de type **amas** : notons **superpixel** les amas et les pixels aux extrémités. On associe chaque *superpixel* un nœud.

D'un point de vue conceptuel, cela se présente par une double association, entre nœud et superpixel. Un **superpixel** est défini par une liste de pixels et le nœud auquel il est associé. La liste de points doit avoir au moins point . Un **nœud** est défini par une liste non vide d'arêtes, une position et le superpixel auquel il est associé.

3.3.2 La recherche des arêtes

Une arête est une ligne curviligne du squelette, elle est limitée par deux superpixels. Les deux superpixels constituent les nœuds de départ et d'arrivée de l'arête associée à la ligne curviligne. Pour trouver toutes les arêtes incidentes à un nœud, nous cherchons toutes les lignes curvilignes partant du superpixel associé au nœud, on parcourt les pixels composant la ligne curviligne jusqu'à tomber sur un superpixel, et le nœud associé à ce superpixel correspond au superpixel du nœud d'arrivée de



(a) (b)

FIGURE 3.7 – Capture des nœuds dans le squelette : (a) squelette entier avec tous les superpixels, (b) gros plan d'une partie de (a)

l'arête associée à la ligne (fig 3.8). On crée donc l'arrête et on l'insère dans les listes d'arêtes des nœuds de départ et d'arrivée. Chaque ligne déjà parcourue (arête créée) est marquée. Nous reprenons le processus pour chaque superpixel pour trouver toutes les arêtes du graphe.

Une arête est définie par une liste non vide de pixels (les pixels de la ligne curviligne associée à l'arête), un nœud de départ, un nœud d'arrivée et un label.

3.3.3 L'ajout d'une arête dans la liste d'un nœud

Lorsqu'une arête est créée, nous l'ajoutons dans les listes de ses deux nœuds incidents suivant un ordre angulaire. Cela nous permet d'ordonner la liste de chaque nœud. En effet, pour ajouter une arête dans la liste d'un nœud, notons $\boldsymbol{n_c}$ (n_c : le nœud courant), nous procédons de la manière suivante : soit la liste est vide, auquel cas, on ajoute l'arête à la première position, soit elle contient au moins une arête, et dans ce cas ; notons $\boldsymbol{a_c}$ (a_c : l'arête courante), l'arête à ajouter, nous calculons β, l'angle de $\boldsymbol{a_c}$ par rapport à la droite horizontale passant par le point de $\boldsymbol{n_c}$. Nous

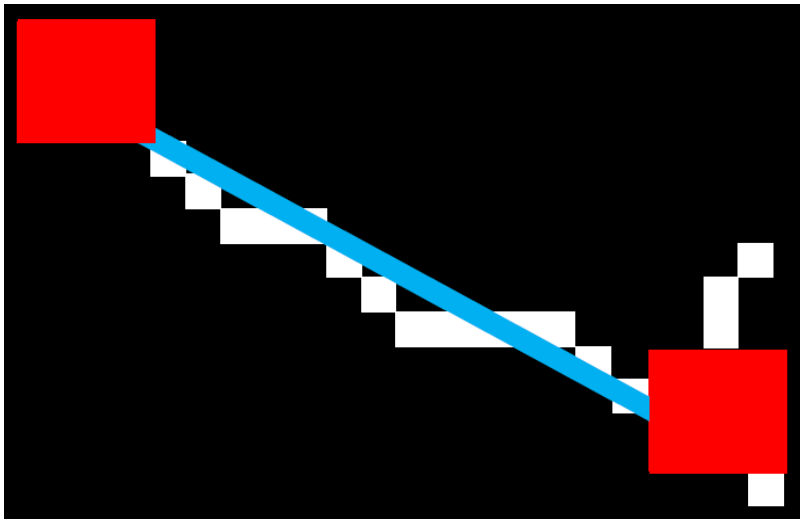


FIGURE 3.8 – Présentation d'une arête : en rouge deux amas associant les nœuds adjacents à l'arête (en bleu), et en blanc, la ligne curviligne représentant une arête du graphe

comparons l'angle un à un avec ceux des arêtes déjà dans la liste et dès que nous tombons sur une arête où l'angle est plus grand que celui de $\boldsymbol{a_c}$, nous ajoutons $\boldsymbol{a_c}$ à la position de cette arête et déplaçons d'une case toutes les arêtes à partir de cette position. Si l'ange β est plus grand que tous les angles des arêtes dans la liste, nous ajoutons $\boldsymbol{a_c}$ à la fin de la liste (fig 3.9).

Nous répétons le processus sur toutes les arêtes dans chaque nœud, et nous générons le graphe du squelette. La figure 3.10 est une représentation géométrique du graphe du squelette de la figure 3.5. Les nœuds en blanc, les arêtes en couleurs variées et le squelette en gris.

3.3.4 Nettoyage du graphe

Nous nettoyons le graphe pour avoir des arêtes plus ou moins alignés. Ce nettoyage est important d'être effectué, mais son intérêt se rapporte à une étape ultérieure du processus de traitement.

Nous avons deux critères pour nettoyer le graphe : le premier critère consiste à retirer toute les arêtes dont la longueur est inférieure à un seuil, et le deuxième consiste à retirer toutes les arêtes fuyantes. Une arête fuyante est une arête dont les pixels de la ligne curviligne qui lui est associée rentre intégralement dans la forme de l'objet (fig 3.11). Nous remarquons dans la figure 3.11 que les lignes curvilignes en rouge sont à l'intérieur de la forme de l'objet. Les arêtes associées à ces lignes

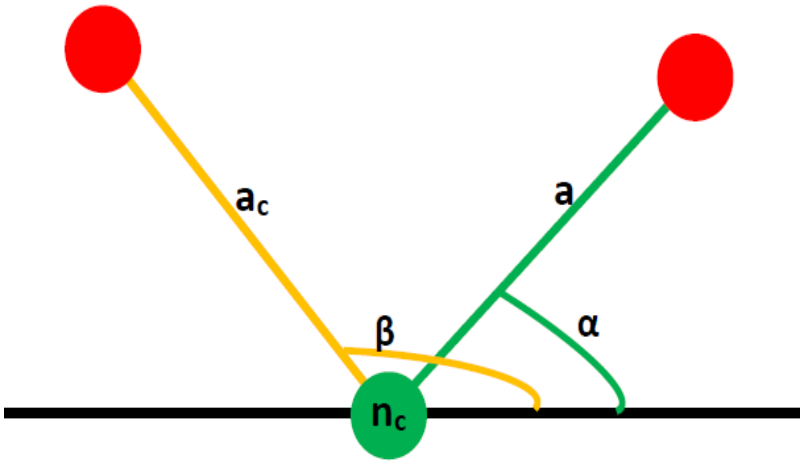


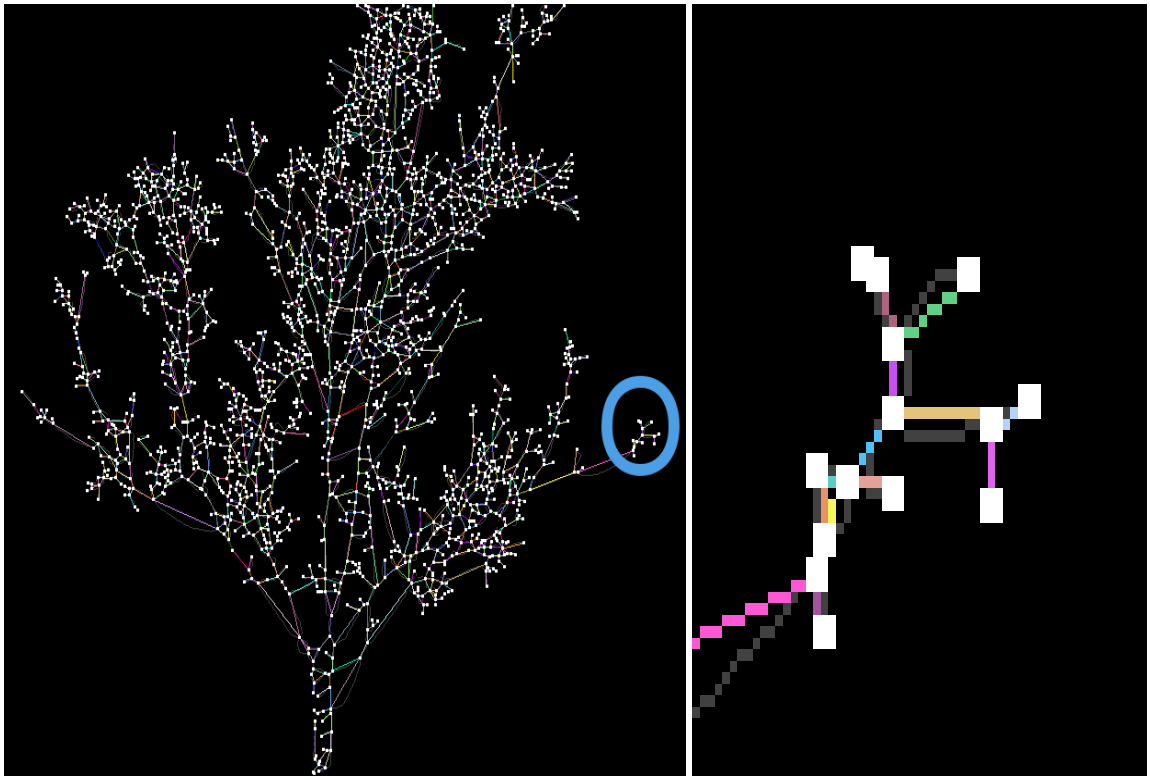
FIGURE 3.9 – Ajout d'une arrête dans la liste d'un nœud
légende : en noir la ligne horizontale passant par le point de $\boldsymbol{n_c}$
en orange ***l'arête*** $\boldsymbol{a_c}$ et $\boldsymbol{a}$ une arête déjà dans liste
dans cette figure, $\beta > \alpha$, implique que $\boldsymbol{a_c}$ soit ajoutée devant $\boldsymbol{a}$,
et donc à la fin de la liste de $\boldsymbol{n_c}$.

sont dites, arêtes incluses.

Par contrainte de temps, nous avons mis en œuvre que le premier critère. Nous déterminons un seuil ε tel que si $l \leq \varepsilon$, avec l la longueur d'une arête. Le choix du seuil est fixé comme suit : nous cherchons la longueur maximale L de toutes les arêtes du graphe et nous prenons 10%, soit $\varepsilon = L{*}10\%$. On suppose donc qu'il n'y a pas d'arête de longueur aberrante : on pourrait utiliser 5% de la longueur médiane (on trie toutes les longueurs par ordre croissant et on prend la valeur centrale).

Nous sélectionnons pour chaque nœud $\boldsymbol{n_c}$, sa liste d'arêtes incidentes, et pour chaque arête $\boldsymbol{a_c}$ dans la liste et de longueur $\boldsymbol{l} \leq \varepsilon$, Nous retirons $\boldsymbol{a_c}$ de la liste, nous ajoutons dans la liste de arêtes de $\boldsymbol{n_c}$ toutes les arêtes de l'autre nœud adjacent à $\boldsymbol{a_c}$, soit $\boldsymbol{m_c}$, excepté $\boldsymbol{a_c}$ et on retire le nœud $\boldsymbol{m_c}$. Puis nous recommençons la sélection, jusqu'à ne plus avoir d'arêtes répondant au critère, ensuite on passe à un autre autre nœud. On réitère le processus jusqu'à couvrir tous les nœuds du graphe.

Le tableau de la figure 3.12 représente quantitativement la différence entre le graphe initial (fig 3.10) et le graphe nettoyé (fig 3.12), il s'agit du graphe du squelette de la figure 3.5



(a) (b)

FIGURE 3.10 – représentation graphique d'un graphe : (a) graphe entier, (b) gros plan de la partie entourée en bleu dans (a)
légende : en blanc les noeuds du graphe $\boldsymbol{n_c}$
en gris le squelette (plus visible dans la figure (b))
dans cette figure, $\beta > \alpha$, implique que $\boldsymbol{a_c}$ soit ajouter devant $\boldsymbol{a}$,
et les arêtes présentées en différentes couleurs (sauf le blanc et le gris)

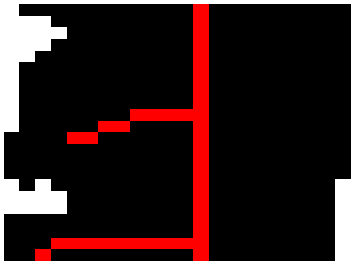


FIGURE 3.11 – Lignes curvilignes d'arêtes incluses

3.3.5 Création des axes

Un axe est une suite d'arêtes (souvent alignées) qui répondent à certains critères. Ces critères peuvent être purement biologiques ou topologiques. Par exemple deux arêtes successives ayant un nœud adjacent en commun, le nœud $\boldsymbol{n_c}$ et avec une angle

FIGURE 3.12 – Graphe nettoyé

graphe	nombre d'arêtes	nombre de nœuds
initial	2956	2235
nettoyé	1590	1224

légende : dans le 2e colonne, le nombre d'arêtes avant et après le nettoyage du graphe, nous avons retiré 1366 arêtes par rapport au graphe initial et dans la 3e colonne, nous avons retiré 1011 nœuds par rapport au graphe initial

α proche de $180°$ par défaut ou par excès, forment ou appartiennent à un axe. Dans la figure 3.13, l'ensemble d'arêtes $\{a_1, a_2, a_3\}$ constitue un axe :
Soit $\boldsymbol{A} = \{a_1, a_2, ..., a_n \,/\, \forall\, a_i, a_j \in$ *à la liste de* $\boldsymbol{n_c}$, $i \neq j$, *on a* $\alpha \simeq 180\ \}$.

Un autre critère serait d'utiliser la distribution du diamètre à chaque pixels d'une ligne curviligne associée à une arête. L'idée est de prendre toujours deux arête successives appartenant à $\boldsymbol{n_c}$ et comparer la variation de diamètre des pixels proches de l'amas associé à $\boldsymbol{n_c}$ pour les deux arêtes. si la variation est faible, nous pouvons dire qu'il s'agit donc d'une même branche, donc les deux arêtes forment ou appartiennent à un axe.

Nous pouvons utiliser les deux critères de façon successive ou la combinaison des deux (la moyenne harmonique des deux critères). Nous n'avons utilisé dans notre pipeline qu'un seul critère, celui qui est à priori le plus significatif (le plus botaniquement parlant). C'est le critère d'angulation. Nous cherchons dans la liste d'arêtes de chaque nœud $\boldsymbol{n_c}$ des couples d'arêtes $\boldsymbol{(a_i, a_j)}$ qui minimisent le plus la

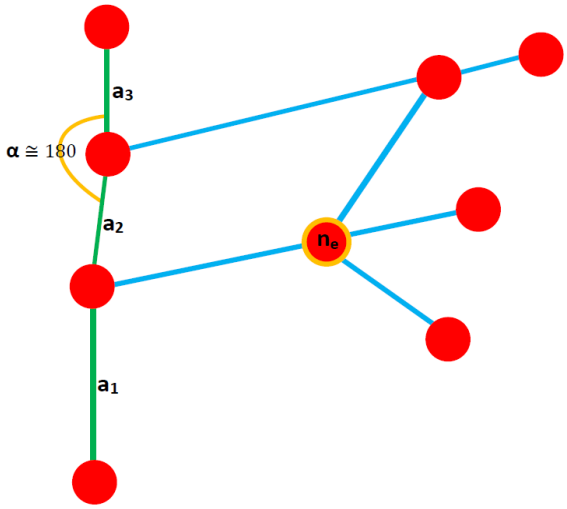


FIGURE 3.13 – Suite d'arêtes formant un axe

variation angulaire : pour chaque couple nous prenons les deux points des nœuds adjacents respectivement à $\boldsymbol{a_i}$ soit **P**, et $\boldsymbol{a_j}$, soit **Q**. Notons **O** le point de $\boldsymbol{n_c}$ (fig 3.14). la variation angulaire $\delta\alpha = |\alpha$ - 180|, où $\alpha = \widehat{POQ}$. Nous avons une relation forte car nous nous assurons aussi que la variation angulaire $\delta\beta = |\beta$ - 180|, où $\beta = \widehat{QOP}$ soit minimale dans ce sens. La réciprocité des alignements implique donc que : $\boldsymbol{a_i}$ et $\boldsymbol{a_j}$ sont alignées si et seulement si, la meilleure candidate de $\boldsymbol{a_i}$ et $\boldsymbol{a_j}$ et si la meilleure candidate de $\boldsymbol{a_j}$ et $\boldsymbol{a_i}$.

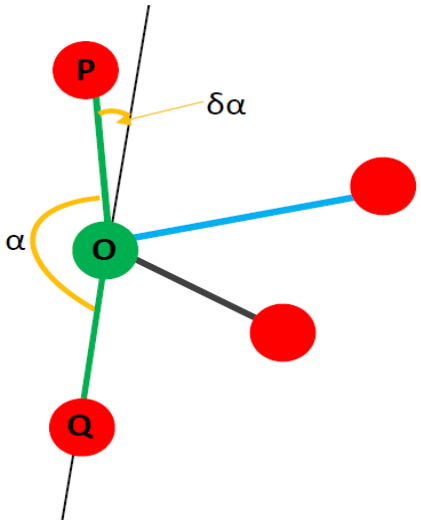
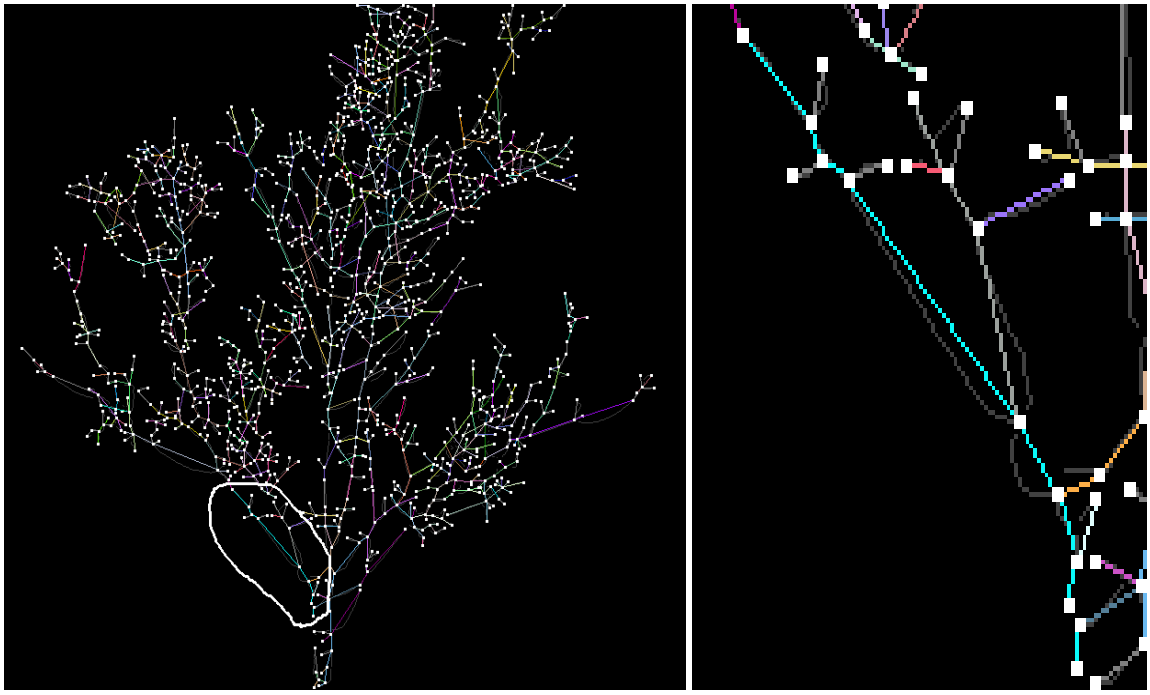


FIGURE 3.14 – illustration du critère d'angulation

Dans la figure 3.14, quand nous aurons attribué un label unique aux arêtes en vert, il en restera deux dans $\boldsymbol{n_c}$, et dans ce cas, quelque ce soit la valeur de leur

variation angulaire, elle sera minimale. Nous avons donc choisi un seuil auquel $\delta\alpha$ ne doit pas dépasser. En effet, deux arêtes qui minimisent la variation angulaire forment ou appartiennent à un axe si $\delta\alpha \leq 20$.



(a) (b)

FIGURE 3.15 – Création des axes : (b) gros plan sur l'axe (en bleu turquoise) de la partie entourée dans (a)

3.4 Passage du graphe à un arbre mathématique

Soit $\boldsymbol{A_{bio}}$, un arbre biologique, $\boldsymbol{A_{bio}}$ n'a pas de cycles, c'est-à-dire qu'une branche n'est rattachée que par l'une de ses extrémités. Or, le graphe que nous avons généré à partir du squelette possède des branches qui se croisent, et font qu'une branche soit portée par plus d'une branche porteuse. Le développement d'un $\boldsymbol{A_{bio}}$ étant dans un espace 3D, alors que nous n'ayons qu'une image 2D de l'arbre, il est évident que le passage d'une forme représentée en 3D en une représentation 2D nous coûte cher. En effet, le perte de l'information de profondeur entraîne des situations qui ne correspondent pas à la réalité. Un cycle dans un graphe est une suite d'arête dont les extrémités sont identiques.

Nous avons donc besoin d'éclater les cycles du graphe pour avoir un arbre mathématique, notons $\boldsymbol{A_{math}}$ un arbre mathématique. Un $\boldsymbol{A_{math}}$ est un graphe sans

cycles, et donc représente mieux un $\boldsymbol{A_{bio}}$.

Le graphe généré est un graphe non orienté. Ainsi, pour trouver les cycles dans le graphe, nous avons créé une nouvelle donnée. Il s'agit de la paire ***(nœud, arête)***. Nous allons pouvoir chercher les cycles présents dans un graphe en simulant la recherche d'une face à partir d'une carte combinatoire[10] comme suit :
Pour chaque arête $\boldsymbol{a_c}$ dans la liste d'arêtes du nœud $\boldsymbol{n_c}$, nous créons la paire $\boldsymbol{(n_c, a_c)}$, on récupère l'arête $\boldsymbol{a_{suiv}}$ qui se trouve juste devant $\boldsymbol{a_c}$ dans la liste d'arêtes du nœud $\boldsymbol{n_e}$, le nœud adjacent à l'autre de bout de $\boldsymbol{a_c}$. Ensuite, nous créons la paire $\boldsymbol{(n_e, a_{suiv})}$ et répétons le processus jusqu'à retomber sur la paire $\boldsymbol{(n_c, a_c)}$. Nous réitérons le processus à partir de $\boldsymbol{a_{prec}}$ qui se trouve derrière $\boldsymbol{a_c}$ dans la listes d'arête du nœud $\boldsymbol{n_e}$. Chaque suite d'arêtes est mise dans une liste qui constitue donc un potentiel cycle, appelé $\boldsymbol{C}$. Pour vérifier que $\boldsymbol{C}$ est un cycle, nous le comparons avec le cycle issu du nœud racine, soit $\boldsymbol{C_{racine}}$. Si $\boldsymbol{C = C_{racine}}$, $\boldsymbol{C}$ n'est pas un cycle, sinon $\boldsymbol{C}$ est un cycle.
Ce mécanisme est possible par ce que nous avons des listes ordonnées d'arêtes. Cela nous a permis en créant la paire *(nœud, arête)* qui me donne une orientation, de palier le fait que nous ayons un graphe non-orienté.

Dans la figure 3.16, Nous pouvons voir en partant de $\boldsymbol{n_c}$, que la paire $\boldsymbol{(n_c, a_c)}$ va trouver en prenant $\boldsymbol{a_g}$, le cycle composé de 4 arêtes en vert (4 pairs de ***(nœud, arête)***) et en prenant $\boldsymbol{a_d}$, le cycle composé de 9 arêtes (13 pairs de ***(nœud, arête)***). Le dernier cycle est égale au $\boldsymbol{C_{racine}}$, et n'est donc pas considéré comme cycle.

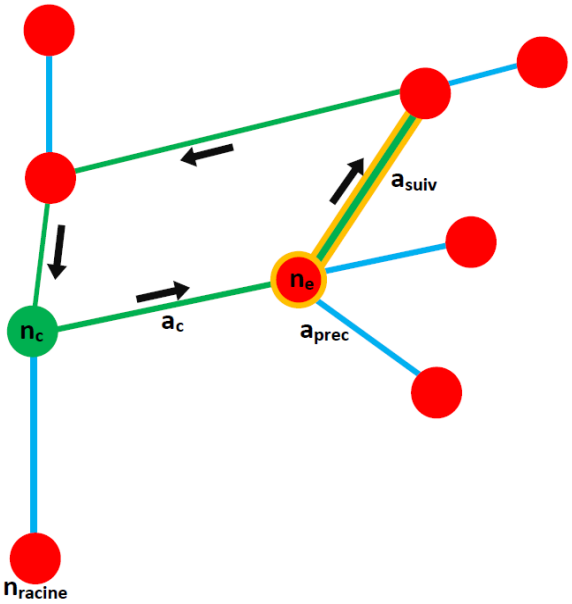


FIGURE 3.16 – Cycle dans un graphe : les arêtes en vert constituent un cycle

3.4.1 Éclatement des cycles

Dans cette partie, Nous présentons la démarche mis en place pour éclater les cycles dans le graphe pour avoir un $\boldsymbol{A_{math}}$. Pour éclater un cycle $\boldsymbol{C}$, nous avons besoin d'ouvrir un nœud (le dupliquer par exemple) ou bien supprimer un nœud. Une arête incidente à ce nœud appartient à $\boldsymbol{C}$. Nous cherchons ce nœud $\boldsymbol{n_c}$, en cherchant une arête $\boldsymbol{a_c}$ dans **le cycle** $\boldsymbol{C}$ que nous appelons arête ***candidate***. Pour trouver $\boldsymbol{a_c}$ dans $\boldsymbol{C}$, nous utilisons des critères de sélection allant du botaniques aux statistiques.

critères de sélection

Les axes : nous cherchons dans $\boldsymbol{C}$, $\boldsymbol{a_c}$ tel que $\boldsymbol{a_c}$ n'appartient pas à un axe. Nous savons que toute arête appartenant à un axe possède un label, alors si une arête n'est pas labellisée, elle devient automatiquement l'arête ***candidate*** (fig 3.17a).
Dans la figure 3.17a, 3 des 4 arêtes constituant le cycle (arêtes incluses dans l'ellipse en bleu) appartiennent à des axes (axe vert, bleu et violet), par conséquent la seule arête qui reste (en gris) devient l'arête ***candidate***. C'est l'un des nœuds adjacents à cette arête qui sera éclaté.
Cet critère présente des limites, car dès que $\boldsymbol{C}$ possède au moins 2 arêtes qui n'ont pas été labellisées (fig 3.17b), il ne sait pas quelle arête choisir ; $\boldsymbol{C}$ peut aussi avoir qu'une seule arête non labellisée, mais qui est un isthme (une arête de $\boldsymbol{C}$ dont un de ses nœuds adjacents n'a qu'une arête dans la liste : fig 3.17c), ou bien toutes les arêtes de $\boldsymbol{C}$ sont labellisées (fig 3.17d). Nous avons donc besoin d'un autre critère de sélection.

Le diamètre : dans ce critère, au lieu de prendre en compte toutes les arêtes de $\boldsymbol{C}$, nous gardons le critère de sélection en fonction des ***axes*** et récupérons la liste de toutes les arêtes de $\boldsymbol{C}$ n'ayant pas de label, s'il y en a (ça nous permet de ne pas sélectionner une arête appartenant à un axe), sinon nous récupérons toutes les arêtes. Nous calculons pour chaque arête $\boldsymbol{a_c}$, le diamètre médian des points des pixels constituant la ligne curviligne associée à $\boldsymbol{a_c}$. et Nous sélectionnons l'arête avec le diamètre le plus petit.

La longueur : le critère diamètre à aussi des limites. Si la différence des diamètres n'est pas significative, ça peut relever du bruit lors de la segmentation ou du calcul du squelette. Dans tous les cas, si la différence ne représente pas une vraie tendance, nous ne pouvons pas choisir. Nous passons à un autre critère, la longueur des arêtes.

Ce critère calcule pour chaque arête a_c de C sa longueur (distance euclidienne entre les points des 2 nœuds adjacents à a_c), soit l. L'arête ***candidate*** est celle qui a la plus petite longueur l.

L'aléatoire : et en dernier, si la méthode de sélection n'arrive pas à trouver une arête ***candidate*** à l'aide des 3 précédents critères, nous choisissons aléatoirement une arête de C.

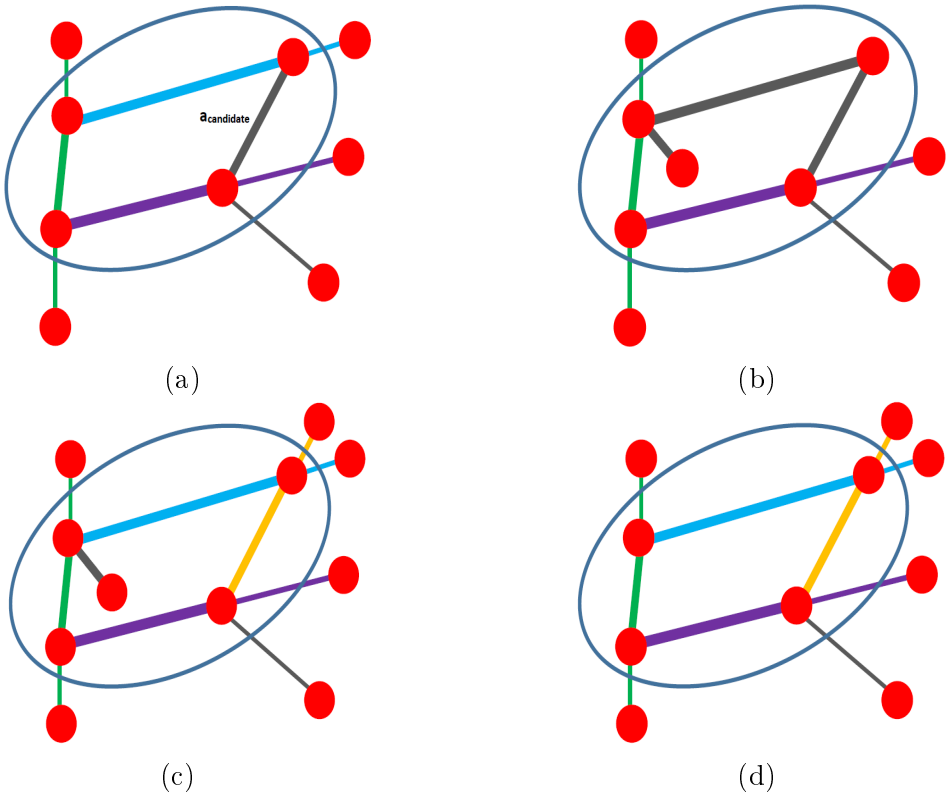


FIGURE 3.17 – Types de cycles

Soit a_c, l'arête sélectionné, pour choisir le nœud à éclater, nous procédons de la manière suivante : nous reprenons le critère de diamètre précédemment défini, nous calculons deux diamètres médians du 1/4 des points proche du nœud de **départ** et d'**arrivé** (nœud de départ et d'arrivé sont adjacents à a_c) des pixels de la ligne curviligne associé à a_c, notons $d_{départ}$ et $d_{arrivé}$ les deux diamètres. Si $d_{départ} < d_{arrivé}$ alors nous sélectionnons le nœud de ***départ***, sinon le nœud d'***arrivé***. Ici encore, si la différence n'est pas significative, nous choisissons un autre critère (exemple, le nœud qui a le moins d'arêtes incidentes entre nœud de ***départ*** et ***arrivé***, ou bien aléatoirement).

Pour éclater le nœud sélectionné, notons n_c, soit toutes les arêtes incidentes à n_c appartiennent toutes à des axes, dans ce cas nous créons une nouvelle arête pour chaque paire d'arêtes appartenant à un même axe, et nous supprimons n_c (fig 3.18a et 3.18b), soit il y a au moins une arête qui n'appartient pas à un axe, dans ce cas, nous créons un nouveau nœud $n_{nouveau}$, nous remplaçons n_c par $n_{nouveau}$ dans a_c et nous supprimons a_c dans n_c (fig 3.18c et 3.18d).

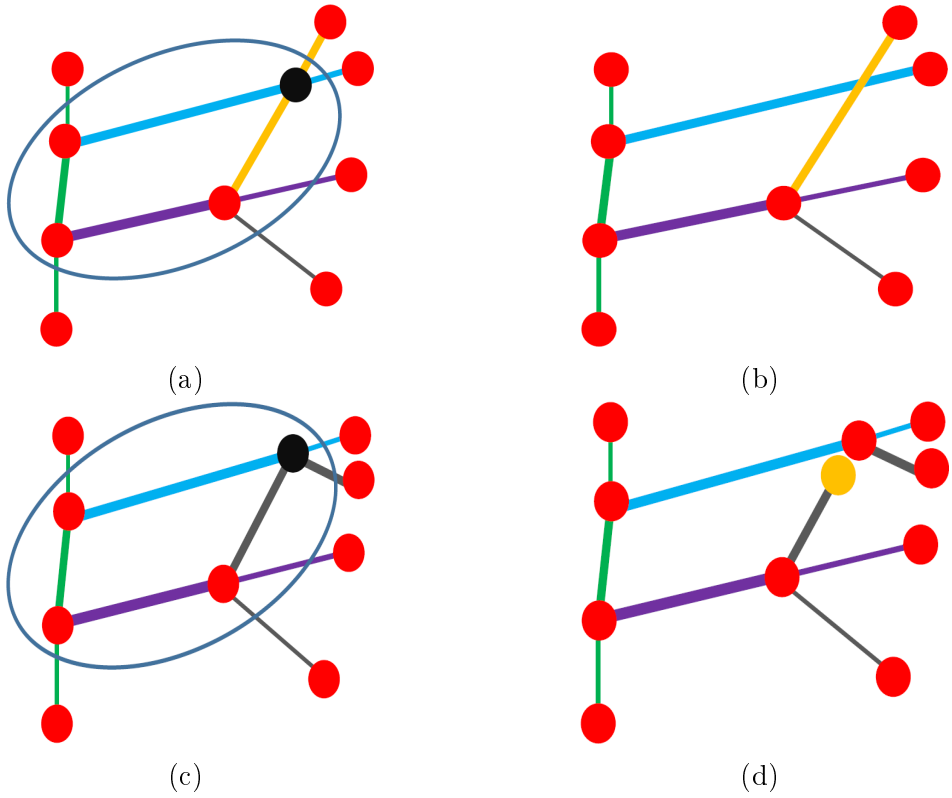


FIGURE 3.18 – Types de cycles
légende : à droite (a et c) les les garphes avec cycle, en noir les nœuds à éclater. À gauche (b et d) les graphes après éclatement des cycles.

Complexité algorithmique

Nous faisons un focus sur la complexité de l'algorithme implémentée dans cette étape. La *recherche/suppression* de cycles n'est pas actuellement optimisée : elle présente une complexité algorithmique en $O(kn)$ où n représente le nombre d'arêtes du graphe. En effet, le processus de suppression des cycles est itératif mais globalement, la détection de tous les cycles nécessite de parcourir une fois chaque arête du graphe dans ses deux sens de parcours. En pratique, il y a un facteur multiplicatif car la

suppression de chaque cycle ne peut pas affranchir l'algorithme de re-parcourir ces arêtes lors de la recherche du cycle suivant. Mais $\boldsymbol{k}$ restera petit au regard de $\boldsymbol{n}$. La recherche de tous les cycles ne s'effectue pas à partir de chaque arête du graphe mais uniquement à partir d'une arête n'ayant pas été atteinte par le parcours d'un cycle précédent.

FIGURE 3.19 – Graphe sans cycle : arbre mathématique

3.5 Reconstruction de l'architecture de l'arbre

Nous présentons dans cette partie, la dernière étape du pipeline. La reconstruction de l'architecture comme il a été défini au tout début de ce document (fig 2.1), consiste à attribuer un ordre de ramification à toutes les branches de l'arbre $\boldsymbol{A_{bio}}$. Nous allons profiter des axes que nous avons crée lors du processus de nettoyage du graphe et de l'arbre $\boldsymbol{A_{math}}$ obtenu après avoir éclater les cycles pour réaliser le ***typage***.

3.5.1 Typage

Le typage consiste à donner un nouveau label identique à tous les axes portés par un autre axe appelé axe porteur. Un axe porté sur $\boldsymbol{A_{math}}$ représente une branche portée dans $\boldsymbol{A_{bio}}$, et un axe porteur dans $\boldsymbol{A_{math}}$ substitue une branche porteuse dans $\boldsymbol{A_{bio}}$. En commençant par l'axe associé au tronc (l'axe dont la première arête à comme nœud de ***départ*** le nœud ***racine***(le germe)), nous attribuons à cet axe,

l'***ordre 0***, noté $\boldsymbol{A_0}$. Ensuite tous les axes portés par $\boldsymbol{A_0}$ vont être d'***ordre 1***. Nous répétons le processus sur tous les axes.

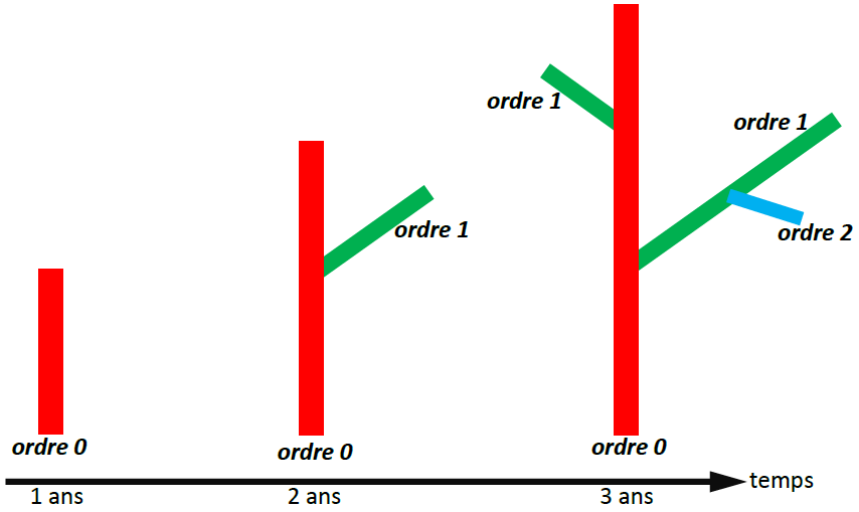


FIGURE 3.20 – Typage des axes

Légende : la reconstruction de l'architecture. ***ordre 0*** représente le tronc de l'arbre, associé à l'axe rouge. Un an plus tard, un bourgeon apical génère une nouvelle branche, associé à l'axe vert d'***ordre 2***. Après trois ans, deux nouveaux axes (***ordre 1*** et ***ordre 2***) apparaissent.

4. Conclusion

4.1 Résultats

Nous avons effectué plusieurs tests afin d'apprécier les résultats de notre méthode. Dans l'annexe (A), nous avons utilisé une image de synthèse (fig 1) et une image couleur d'un vrai arbre. Les résultats que nous présentons sont réalisés sur un ordinateur portable (intel Core i5, 2.70GHz(4CPU), 8Go de RAM, système d'exploitation : Windows 7).

Avec l'image de synthèse (taille : 1.08 Mo, résolution 1200 x 946), nous avons eu des résultats très satisfaisants (fig 2), avec une durée d'exécution faible (26.6 secondes au totale).

Action exécutée	**temps d'exécution**
Passage du squelette au graphe	26.44 sec
Nettoyage du graphe	0.01 sec
Création des axes	0.0033 sec
Passage du graphe à $\boldsymbol{A}_{\boldsymbol{math}}$	0.05 sec
Typage	0,00071 sec

Pour l'image couleur d'un vrai arbre de la figure 1.1 (taille : 132 Ko, résolution 742 x 710), nous avons des résultats plutôt encourageant, en tenant compte de la complexité de la forme de l'arbre. En revanche, le temps d'exécution est élevé par rapport à l'image synthétique, dans certaines étapes du processus.

Action exécutée	**temps d'exécution**
Passage du squelette au graphe	47,42 sec
Nettoyage du graphe	0.14 sec
Création des axes	13.14 sec
Passage du graphe à $\boldsymbol{A}_{\boldsymbol{math}}$	4h40min31.74 sec
Typage	0,04 sec

4.2 Discussions

L'objectif du travail a été atteint, tout en restant sujet à des améliorations. Nous avons la possibilité de générer un graphe à partir d'une image d'arbre. Nous pouvons réaliser un nettoyage basé sur les longueurs des arêtes du graphe, ainsi que former des axes, transformer le graphe en arbre mathématique et enfin reconstruire l'architecture de l'arbre.

Toutefois, comme nous pouvons le constater dans les résultats (Annexe A) de l'image couleur du vrai arbre, notamment sur la création des axes, nous nous apercevons que le critère utilisé (critère angulaire) ne suffit pas. Le passage du graphe à l'arbre mathématique devient très gourmand en terme de temps d'exécution, lors de l'étape de la *détection/suppression* des cycles.

Nous avons présenté une approche automatique permettant de reconstruire l'architecture d'un arbre à partir d'images numériques 2D. Ce qui serait intéressant, c'est de revoir les critères de création des axes et/ou d'éclatements de cycles et d'en améliorer (exemple en utilisant l'information couleur des branches), ou bien changer de données d'entrées, comme les images 3D ou la stéréo-vision pour avoir l'information de profondeur. Du point de vue implémentation, les algorithmes implémentées ne sont pas forcément optimisés. Ces derniers n'étant pas un objectif, ils peuvent être à sujet de perspectives en révisant les algorithmes ou en parallélisant le traitement.

4.3 Apports

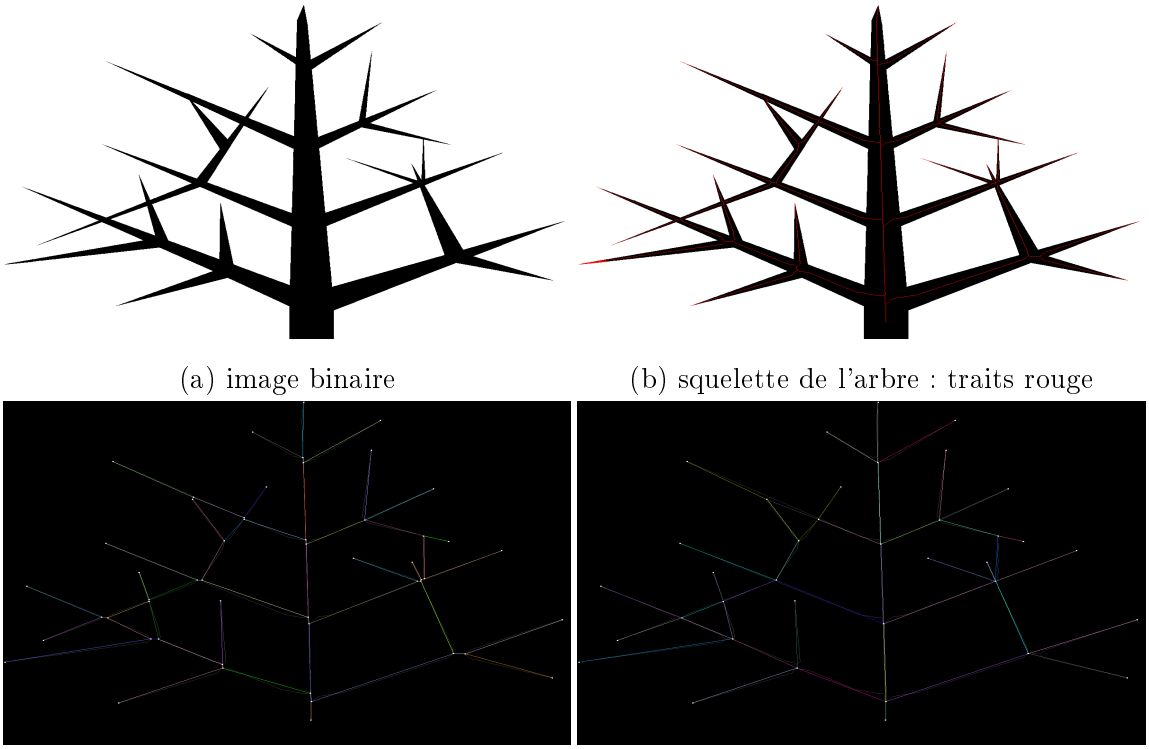
Ce stage de recherche et développement m'a permis de mieux comprendre le fonctionnement d'un travail de recherches et mon intérêt pour l'analyse et le traitement d'images. Par ailleurs, nous avons eu la chance de faire des sorties terrains pour prendre nous même certaines photos pour notre travail. Nous avons pu nous rendre compte à quel point il était difficile de réaliser des acquisitions satisfaisantes. Notre travail de stage s'insère dans un projet dont la finalité est d'aider les botanistes dans la phase de diagnostic d'un arbre.

Annexes

A Résultats

Image synthétique

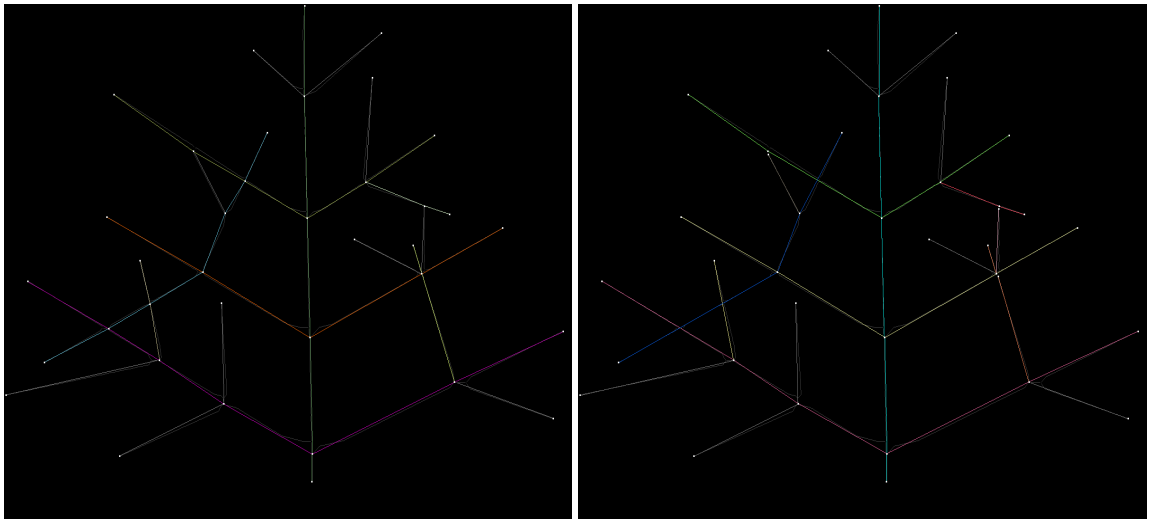
nous avons choisi de réaliser un test de l'application sur une telle image pour les particularités suivantes : dessiner des axes coniques pour simuler le comportement botanique des branches, s'affranchir des accidents de la squelettisation et créer différents contextes d'intersection pour étudier la suppression des cycles.



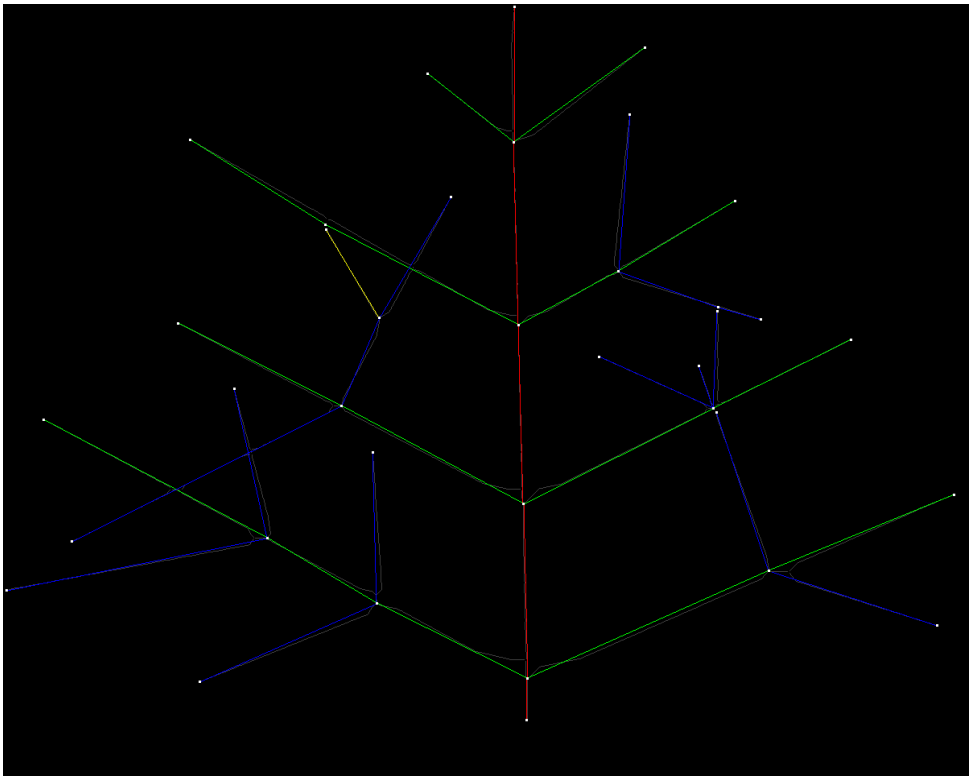
(a) image binaire (b) squelette de l'arbre : traits rouge

(c) graphe non nettoyé (d) graphe nettoyé

FIGURE 1 – Résultat de l'exemple 1.a



(a) création des axes (b) éclatement des cycles



(c) reconstruction de l'architecture

FIGURE 2 – Résultat de l'exemple 1.b

graphe	**nombre d'arêtes**	**nombre de nœuds**
initial	58	52
nettoyé	42	37

• nombre de points dans le squelette : 6456
• nombres de cycles : 6

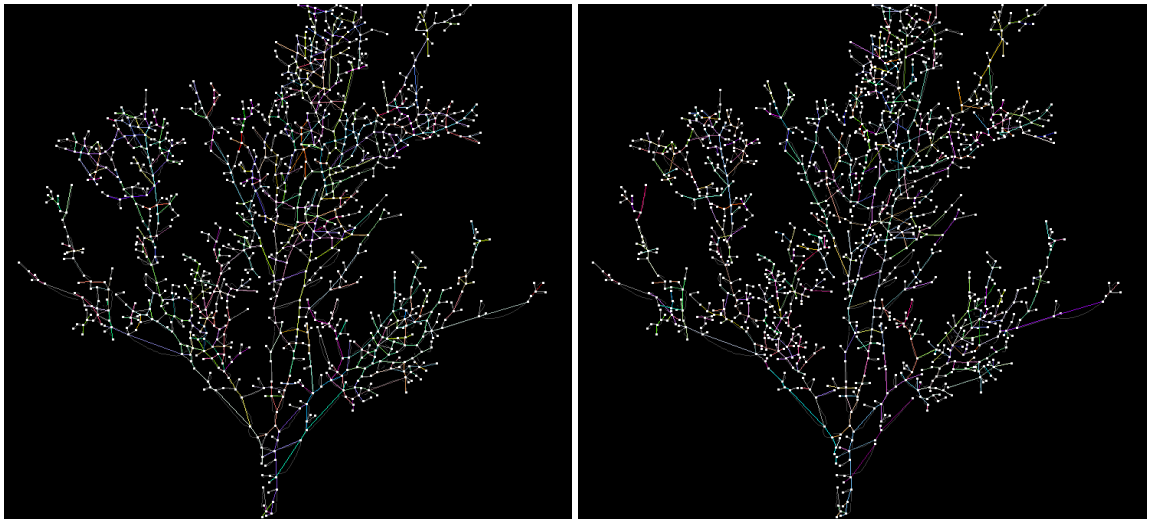
Image d'une vraie arbre

Nous reprenons l'image utilisée pendant la synthèse (fig 1.1). L'enchainement des étapes est présenté comme suit :

- Passage de l'image couleur en niveau de gris et choix du canal (fig 3.1)
- Segmentation de l'image (fig 3.2b)
- Squelettisation de l'image binaire (fig 3.5)
- Passage du squelette au graphe (fig 3.10)
- Nettoyage du graphe par suppression de petites arêtes (fig 3.12)

graphe	**nombre d'arêtes**	**nombre de nœuds**
initial	2956	2235
nettoyé	1590	1224

• nombre de points dans le squelette : 23383
• nombres de cycles : 428



(a) création des axes (b) éclatement des cycles

(c) reconstruction de l'architecture

FIGURE 3 – Résultat de l'exemple 1.b

B Algorithmes

B.1 Passage du squelette au graphe

Données : ***points*** : *liste de points des pixels de branchements*;
pour *point* ∈ *points* **faire**
    si *le point n'a pas été traité* **alors**
        créer une liste et ajouter le *point*;
        mettre dans la liste tous les voisins de *point* de pixel nœud;
        marquer le pixel et ses voisins comme traité;
        créer un *superpixel* avec la liste créée;
    fin
fin

Algorithme 1 : fonction pour la création des superpixels

Données : ***superpixels*** : *liste de superpixels des nœuds du graphe*;
pour *superpixel* ∈ *superpixels* **faire**
    points ← récupérer les points des pixels voisins au *superpixel*;
    pour *point* ∈ *points* **faire**
        créer une liste et ajouter *point*;
        tant que *le pixel de point est un pixel d'arête* **faire**
            point ← le point du pixel voisin suivant;
            marquer le pixel comme traité;
            ajouter *point* dans la liste;
        fin
        si *point est un pixel nœud* **alors**
            récupérer le superpixel où ∈ *point*;
            créer une *arête* de nœud de départ, le nœud courant et de nœud d'arrivé le nœud du superpixel récupéré;
        fin
    fin
fin

Algorithme 2 : fonction pour la création des arêtes

Données : ***noeuds*** : *liste des nœuds du graphe*;
i ← 0 ;
tant que *la liste des noeuds n'est pas finie* **faire**
    noeud ← *noeuds[i]*;
    pour *chaque arête* ∈ *arêtes incidentes au noeud* **faire**
        //le noeud de départ est le noeud courant;
        si *la longueur de l'arête* < *10% longueur max* **alors**
            enlever l'*arête* dans la liste des arêtes des noeuds *départ* et *arrivé*;
            ajouter les arêtes du noeud *arrivé* dans la liste des arêtes du noeud *départ*;
        fin
    fin
    si *il y a eu suppression d'arête* **alors**
        i ← 0 //reprendre à 0;
    sinon
        i ← *i+1*;
    fin
fin

Algorithme 3 : fonction pour le nettoyage du graphe

```
Données : noeuds : liste des nœuds du graphe;
pour chaque noeud ∈ noeuds faire
    récupérer les couples (a_i, a_j) qui répondent au critère d'alignement;
    pour chaque couple ∈ couples faire
        si a_i et a_j n'ont pas de label alors
            créer un nouveau label et l'attribuer à a_i et a_j;
            passer au couple suivant;
        fin
        si un des deux a un label alors
            attribuer le label de l'arête labellisée à l'arête non labellisée;
            passer au couple suivant;
        fin
        si ils ont deux labels différents alors
            choisir un des deux labels, et changer toutes les arêtes ayant l'autre label;
            passer au couple suivant;
        fin
    fin
fin
```

Algorithme 4 : fonction pour la création des axes

B.2 Passage du graphe à un arbre mathématique

```
Données : noeuds : liste des nœuds du graphe;
pour chaque noeud ∈ noeuds faire
    récupérer la liste d'arêtes incidentes à noeud;
    pour tout arête ∈ dans la liste d'arêtes faire
        créer le couple (noeud, arête);
        chercher le cycle C en prenant l'arête suivant arête dans la liste d'arêtes du noeud d'arrivée de l'arête;
        si C est un cycle alors
            chercher dans C l'arête à éclater;
            chercher le noeud n adjacent à arête à éclater;
            éclater n chercher le nouveau cycle à partir du noeud racine C_racine;
        fin
        chercher le cycle C en prenant l'arête précédant arête dans la liste d'arêtes du noeud d'arrivée de
          l'arête;
        si C est un cycle alors
            chercher dans C l'arête à éclater;
            chercher le noeud n adjacent à arête à éclater;
            éclater n chercher le nouveau cycle à partir du noeud racine C_racine;
        fin
    fin
fin
```

Algorithme 5 : fonction pour la suppression des cycles

B.3 Reconstruction de l'architecture

```
Données : noeuds : récupérer la liste des nœuds qui ∈ à l'axe du tronc;
          noeuds' ← noeuds;
          noeuds'' ← lire() // une liste vide de noeuds;
          ordre ← 1 //ordre de ramification zéro (ordre 0) étant le tronc;
pour tout noeud ∈ noeuds' faire
    arêtes ← liste des arêtes incidentes à noeud;
    pour arête ∈ arêtes faire
        si arête ∉ axe alors
            //arête ∉ axe, mais est ramifié avec un noeud du tronc, donc c'est un potentiel axe composée
              d'une seule arête;
            arête.ordre ← ordre;
        sinon
            arête.ordre ← ordre;
            récupérer l'axe où ∈ arête;
            donner à cet axe ordre;
            insérer dans noeuds'' tous les noeuds qui ∈ à axe;
        fin
    fin
    si tous les noeuds de noeuds' ont été parcourus et noeuds'' n'est pas vide alors
        ordre ← ordre + 1;
        ajouter noeuds'' à la fin de noeuds';
        vider noeuds''
    fin
fin
```

Algorithme 6 : fonction pour le typage

C Squelettisation

C.1 Expérimentations

Nous avons comparé l'implémentation de la méthode de [8] dans *ImageJ* avec deux autres implémentations. L'une des deux est de la famille des méthodes de squelettisation par érosion, c'est celle de [11] et l'autre, celle de [7] fait partie de la famille de l'évolution continue de bords des objets. Le choix de comparer avec ces deux méthodes se justifie du fait que :

- [11], en regardant les aspets algorithmiques, la méthode d'*ImageJ* s'apparente à une optimisation de cette dernière
- [7] appartient à une autre famille, ce qui nous permettra d'avoir assez d'éléments pour comparer les résultats.

Les expérimentations réalisées portent sur les différences au niveau de la reconnexion des composantes.

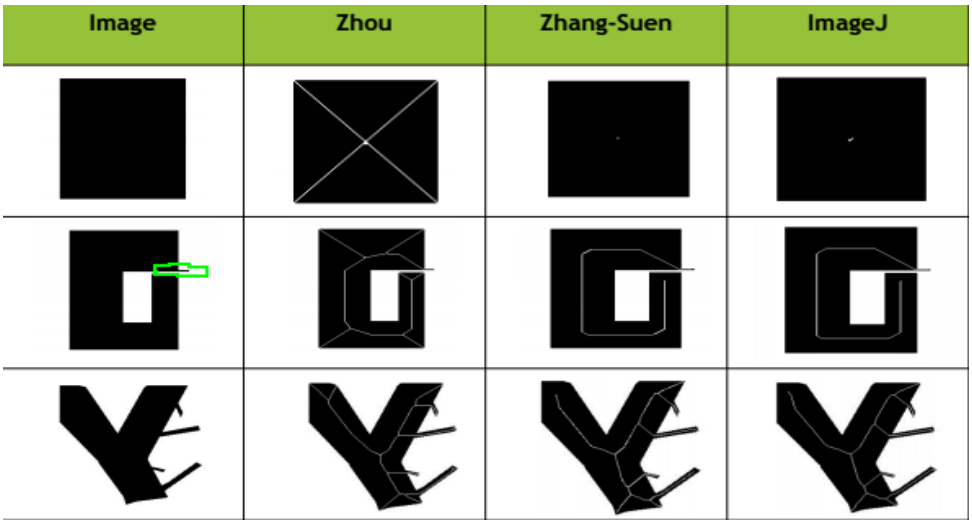


FIGURE 4 – Comparaison des trois méthodes sur 3 images différentes : un carré, un objet en forme de G avec une petite ligne représentant un artefact et une forme représentant un branchement d'axes.

C.2 Précision algorithmique

L'algorithme proposé par [11] repose sur le calcul de deux paramètres :

1. le nombre d'alternances noir/blanc et

2. le nombre de points noirs dans le *8-voisinage* d'un pixel donné.

Les valeurs de ces paramètres sont comparés à des configurations prédéfinies (moins de 20 tests).
ImageJ propose une autre alternative d'implémentation en définissant les points à enlever et/ou garder dans le mécanisme d'érosion au regard des 256 combinaisons décrivant des arrangements possibles des points noirs et blancs autour d'un point donné.
L'algorithme de [7] utilise à la fois une carte de drapeau et l'image pour la suppression des pixels de frontière. La carte est utilisée pour signaler les pixels qui seront supprimés à la fin.

Conclusion : Intuitivement, la méthode d'*ImageJ* est globalement identique à celle de [11, T.Y. Zhang et al], seule l'implémentation est différente. Cette implémentation conduit à des temps de calcul courts pour un résultat sensiblement identique. l'implémentation de *ImageJ* a bien de comportement normal au regard de la famille d'algorithme à laquelle elle appartient.
En revanche, il est vital de nettoyer préalablement l'image (lissage, suppressions des trous, des barbules ...) afin de produire des squelettes présentant un minimum d'axes de "*raccord*", voir quelques illustrations dans l'annexe C.

L'intérêt de nettoyer préalablement l'image permet de réduire les raccords et produit des squelettes assez plus proche de la forme de l'objet. La figure 5 est un arbre sans feuille, prise près du collège Castelnau le 19 mars 2017.

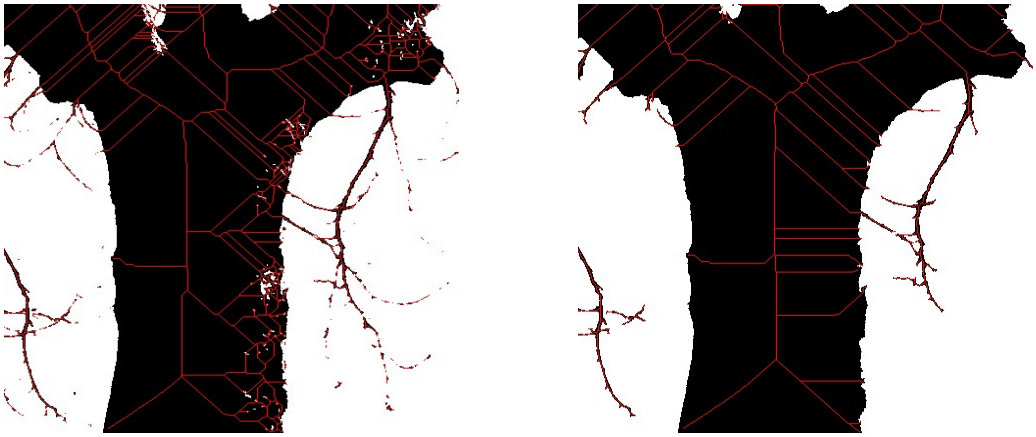
La figure 6 est une squelettisation du canal bleu, après segmentation de l'image de la figure 5. Les pixels en couleur rouge représentent le squelette, les pixels en couleur noir, l'objet et le fond en blanc.

Nous avons, ensuite effectuer un nombre fini d'érosion, afin d'évaluer la différence, et l'intérêt de nettoyer l'image (fig 7). dans la figure 8, nous faisons un gros plan sur le tronc des deux squelettes, et remarquons toute de suite l'intérêt de nettoyer l'image. Cela vient consolider l'argument selon lequel le comportement de la méthode de squelettisation d'*ImageJ* n'est pas aberrante et représente un comportement normal.

FIGURE 5 – Image d'arbre en couleur.

FIGURE 6 – Squelette avant nettoyage de l'image

FIGURE 7 – Squelette après nettoyage de l'image (érosion multiple)



(a) (b)

FIGURE 8 – Gros plan des squelettes au niveau du tronc : (a) le squelette avant nettoyage de l'image et (b) le squelette après nettoyage de l'image

Bibliographie

[1] A. Lindenmayer, J. S.Hanan, F. D. Fracchia, D. R. Fowler, M. J. D. Boer, and L. Mercer, *The Algorthmic Beauty of Plants*. 1996.

[2] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan, "Single image tree modeling," 2008.

[3] B. Neubert, T. Franken, and O. Deussen, "Approximate image-based tree-modeling using particle flows," 2007.

[4] Z. Wang, L. Zhang, T. Fang, X. Tong, P. Mathiopoulos, L. Zhang, and J. Mei, "A local structure and direction-aware optimization approach for three-dimensional tree modeling," 2016.

[5] J. Diener, P. Nacry, C. Périn, A. Diévart, X. Draye, F. Boudon, A. Gojon, B. Muller, C. Pradal, and C. Godin, "An automated image-processing pipeline for high-throughput analysis of root architecture in openalea," 2013.

[6] N. Otsu, "A threshold selection method from gray-level histograms," 1979.

[7] R. W. Zhou, C. Quek, and G. S. Ng, "A novel single-pass thinning algorithm and an effective set of performance criteria," 1994.

[8] T.-C. Lee and R. L. Kashyap, "Building skeleton model via 3d medial/axis thinning algorithms,"

[9] P.-E. Danielsson, "Euclidian distance mapping," 1980.

[10] A. Vince, "Combinatorial maps," 1981.

[11] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," 1984.