Vectorisation du cadastre ancien : restructuration de la chaîne de traitement, implémentation d'une nouvelle méthode de détection et utilisation de la théorie des graphes

Anthony Chalais

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

ECOLE SUPERIEURE DES GEOMETRES ET TOPOGRAPHES

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : Géomètre et Topographe

par

Anthony CHALAIS

Vectorisation du cadastre ancien : restructuration de la chaîne de traitement, implémentation d'une nouvelle méthode de détection et utilisation de la théorie des graphes

Soutenu le 06 Septembre 2019

JURY

Monsieur Jean-Michel FOLLIN	Président du jury et maître de stage
Madame Elisabeth SIMONETTO	Maître de stage
Monsieur Vincent HABCHI	Enseignant référent

Remerciements

Je souhaite d'abord remercier mes deux maîtres de stage, Mme Elisabeth SIMONETTO et M. Jean-Michel FOLLIN, qui ont su m'accompagner et être patients tout au long de mon TFE, lors de l'élaboration de la chaîne de traitement et pour la rédaction de ce mémoire.

Ensuite, je remercie mon professeur référent, M. Vincent HABCHI, qui m'a particulièrement aidé lors de problèmes de programmation, mais aussi quant à la rédaction de ce mémoire.

Je remercie aussi M. Frédéric DURAND, pour son assistance sur l'utilisation du serveur de calcul du laboratoire.

Je remercie également M. Pierre TOUZARD, qui a su me montrer et me conforter sur l'importance d'une telle chaîne, et pour son apport de données pour les futurs essais de la chaîne de traitement.

Je remercie mes camarades de l'école, et en particulier ceux de la salle T0 et du groupe de TP de 2ème année, pour leur convivialité, leurs réconforts et leurs conseils durant ces cinq mois de TFE.

Enfin, je souhaite remercier mes parents, ma grand-mère, mon frère et ma belle-sœur pour qui, les moments parfois difficiles et la distance liés aux études n'auront pas été un frein à leur soutien moral et financier, indéfectibles tout au long de mes études.

Pour terminer, je souhaite dédier ce mémoire, point d'orgue de mes études mancelles, à mes grands-parents maternels, qui m'auront vu partir au Mans, mais pas revenir malheureusement. Ils auront su me donner la force dont j'avais besoin pour arriver à cet aboutissement.

Liste des abréviations

TFE : Travail de Fin d'Études

PPP : Projet Pré-Professionnel

GeF : Géomatique et Foncier

DGI : Direction Générale des Impôts

DGFiP : Direction Générale des Finances Publiques

PCI : Plan Cadastral Informatisé

LSD : *Line Segment Detector*

HP : Transformée de Hough Probabiliste

NFA : Number of False Alarms

PFA : Probabilité de Fausses Alarmes

PBD : Probabilité de Bonnes Détections

Glossaire

Un **logiciel libre** est un logiciel dont l'utilisation ou la modification peut se faire librement et gratuitement. La seule limite est la licence qui régit les droits d'utilisation et de modification. C'est l'opposé du logiciel propriétaire.

Le **seuillage par Hystérésis** est un procédé utilisé dans le filtrage d'une image. Il binarise l'image en utilisant un seuil bas et un seuil haut souvent différent. Tout ceci implique une non-linéarité. Il permet de traiter des images bruitées, en donnant la couleur noire à un pixel correspondant potentiellement à du bruit, et en donnant la couleur blanche pour les autres pixels par exemple.

Table des matières

Introduction

Depuis le XIX^{ème} siècle, le cadastre français est considéré comme un héritage important du pays. Au fil des années, il a sans cesse évolué. Aujourd'hui, les plans sont plus précis et plus simples d'utilisation, même s'ils sont parfois peu mis à jour. De plus, des travaux de vectorisation de l'ensemble du parcellaire français ont été entrepris, ce qui aide à la formation du PCI-Vecteur. Néanmoins, cette vectorisation est réalisée sur plusieurs années, puisque la seule technique existante repose sur un traitement presque intégralement manuel.

Depuis 2014, des recherches visant à étudier l'évolution du territoire au travers de son parcellaire cadastral sont menées au laboratoire Géomatique et Foncier (GeF). Celles-ci nécessite une base de données vectorielle du parcellaire. Cela implique de vectoriser l'ensemble des planches existantes du cadastre, ce qui est impossible à réaliser en mode entièrement manuel. Depuis 2014, une chaîne de traitement semi-automatique est en cours d'élaboration. Plusieurs travaux d'étudiants se sont succédés (un PPP en 2014, le TFE de Maïté Fahrasmane en 2016 (1), le TFE de Charlotte Odie en 2017 (2), et le TFE de Jean-Marc Beveraggi en 2018 (3)) pour développer chaque étape de la chaîne.

À la suite de ces travaux, le processus est décomposé en plusieurs « moulinettes », n'utilisant que des logiciels libres (Python, QGis, GRASS...), hormis ENVI pour réaliser les masques des planches (mais il est simple d'y remédier). La manipulation des logiciels libres et open-source permet de gérer les étapes du processus, sans avoir besoin d'utiliser des « boîtes noires »[1]. Néanmoins, il serait souhaitable de limiter la chaîne de traitement à un seul logiciel libre (en l'occurrence Python), car la multiplication des logiciels nuit à la lisibilité et l'utilisation. À cela, s'ajoutent de réelles difficultés liées à la vectorisation (limites mal détectées, traitement lourd et difficilement personnalisable). Ce rapport va donc tenter de répondre à différentes problématiques : **Comment mieux structurer le programme ? Comment perfectionner l'étape de vectorisation ? Que peut apporter la théorie des graphes ?**

[1] Terme définissant les logiciels dont on n'a pas accès au code source, et dont le processus n'est pas clairement identifiable.

Nous commencerons par présenter la chaîne de traitement existante et les améliorations que nous y avons apportées, notamment la réorganisation du code. La seconde partie traitera de l'étape de vectorisation. Cette étape étant la plus importante du processus, nous avons cherché à l'améliorer avec un autre algorithme de détection : le *Line Segment Detector* (4). Dans un troisième temps, nous discuterons des améliorations possibles de la chaîne par l'insertion d'un outil puissant et reconnu : la théorie des graphes. Nous verrons que différentes utilisations sont possibles. Enfin, nous mettrons en œuvre notre méthode et évaluerons la qualité des résultats obtenus, avant de conclure sur les possibles perspectives de perfectionnement de la chaîne.

I La chaîne de traitement

I.1 Chaîne au début de ce TFE

I.1.1 Présentation de la chaîne

Afin d'obtenir des parcelles vectorisées à partir de planches, trois étapes sont nécessaires (voir Figure 1) : la vectorisation, le géoréférencement et le mosaïquage. L'enchaînement choisi dans le cadre des travaux du GeF est différent de celui utilisé généralement dans les projets similaires, où le traitement commence par le géoréférencement.

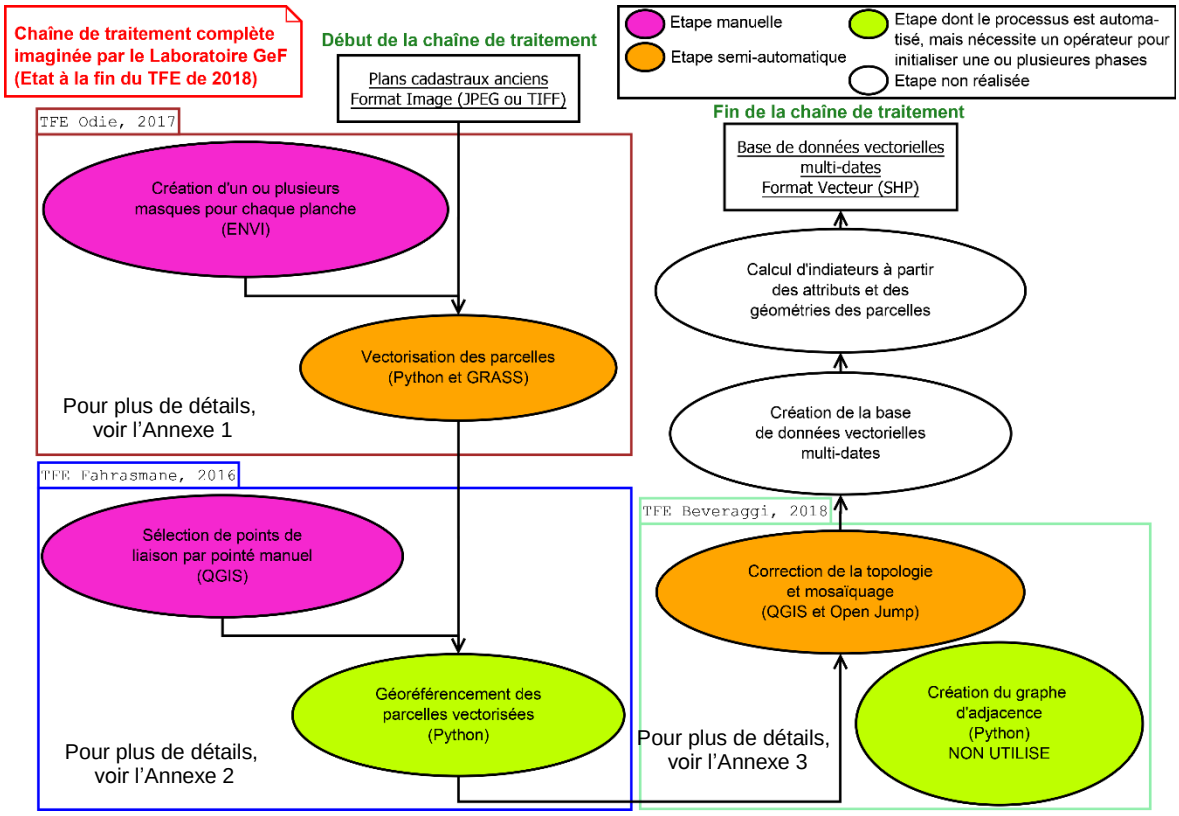


Figure 1 : Chaîne complète imaginée par le Laboratoire GeF, d'après Fahrasmane (2016)

Cette solution est basée sur une idée principale : la non-déformation des limites. En effet, le géoréférencement des planches transforme nécessairement certaines droites en courbes. Or, les parcelles ont été principalement dessinées avec des limites droites, pour simplifier le repérage sur le terrain grâce aux bornes. Commencer par la vectorisation des parcelles permet de garder des limites rectilignes, même si cette étape demeure la plus complexe à mettre en œuvre en raison de la multiplicité des cas et du bruit. Cela aura d'ailleurs pour conséquence de majorer les erreurs aux étapes suivantes.

I.1.2 Les développements réalisés dans les projets précédents

Le premier projet, un PPP de 2014, a démontré l'utilité d'une telle chaîne. Les membres du projet ont vectorisé manuellement des parcelles identifiées sur des planches scannées. Ces données ont ensuite servi aux travaux de géoréférencement.

La première grande étape de la chaîne de traitement a été réalisée par Maïté Fahrasmane, lors de son TFE de 2016 (1). Elle a conçu et testé plusieurs algorithmes permettant le géoréférencement de données vectorielles à partir de points d'appui déterminés manuellement. Les statistiques obtenues à l'issue du stage ont permis de sélectionner le meilleur algorithme : la méthode de régression « ridge » à noyau gaussien[2]. Le schéma fonctionnel du processus de géoréférencement mis en place est détaillé en Annexe 2.

L'étape la plus complexe a été traitée par Charlotte Odie dans son TFE de 2017 : la vectorisation (2). C'est la première opération réalisée dans la chaîne de traitement. La complexité du traitement, l'impossibilité d'obtenir des résultats définitifs (qui auraient permis d'évaluer les conséquences de l'étape sur le reste de la chaîne), et le manque de temps, ont abouti à des données hétérogènes en termes de qualité, et à un processus mêlant plusieurs programmes, qui empêche le déroulement d'un seul tenant de cette partie de la chaîne. Le schéma fonctionnel du processus de vectorisation adopté se trouve en Annexe 1.

Enfin, la dernière étape de la chaîne, le mosaïquage, a été abordée par Jean-Marc Beveraggi dans son TFE de 2018 (3). Il a proposé un processus presque complet de correction topologique (voir le schéma fonctionnel en Annexe 3), et a entamé les recherches sur l'utilisation de la théorie des graphes pour cette opération. Néanmoins, sa chaîne utilise également plusieurs programmes, qui corrigent les éléments sortis de la vectorisation dans le mosaïquage. En effet, la non-correspondance des extrémités à l'issue de la vectorisation et du géoréférencement majore le nombre d'erreurs topologiques à corriger, ce qui requiert un processus complexe.

[2] Dans la suite, nous utiliserons uniquement cette méthode de géoréférencement. Néanmoins, dans un objectif de personnalisation complète de la chaîne de traitement, le processus devrait pouvoir rendre possible l'utilisation d'un autre algorithme de géoréférencement.

I.2 Restructuration en un programme

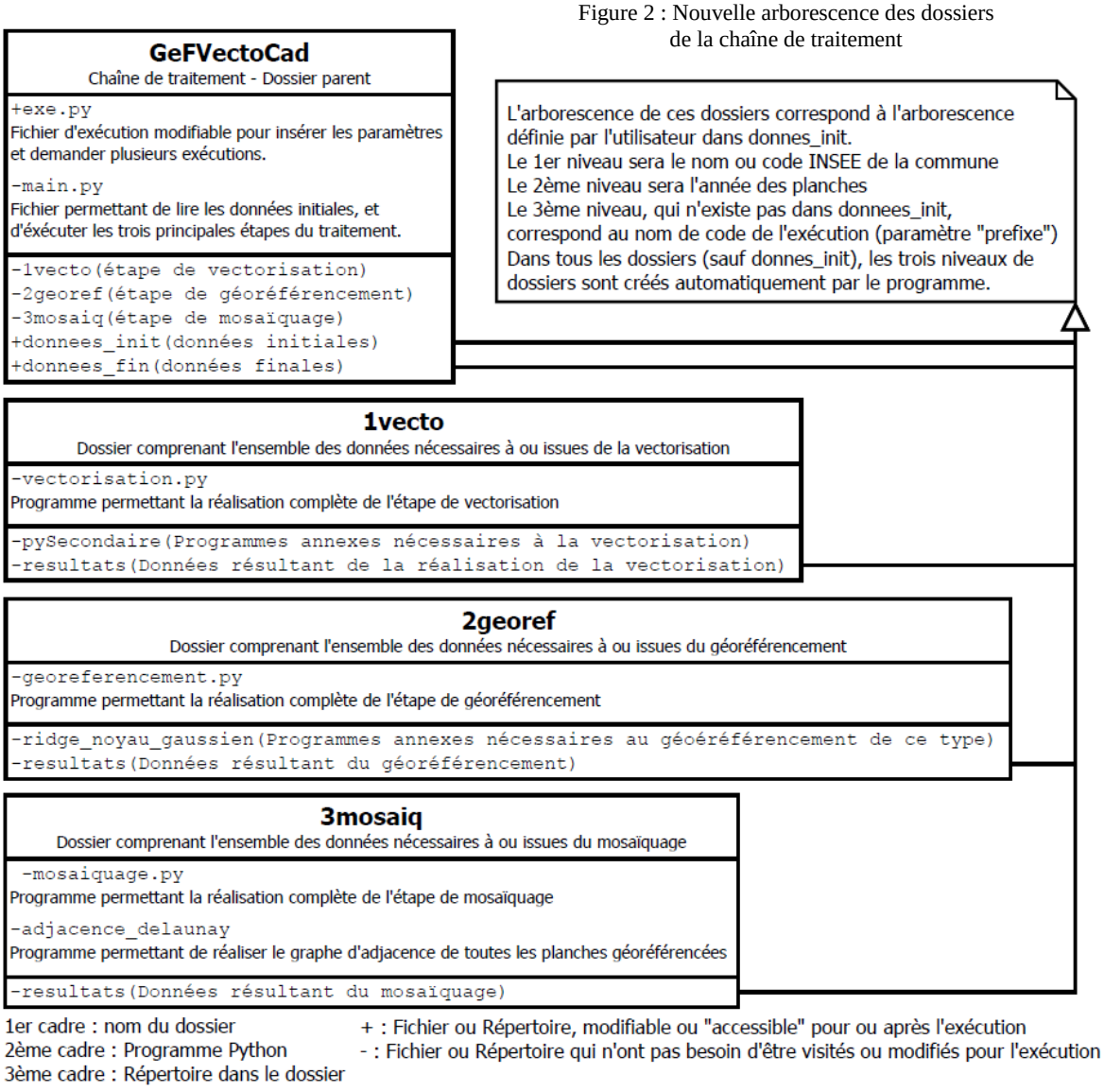
Comme expliqué précédemment, chaque projet s'est focalisé sur une étape de la chaîne. Cependant, la distribution des fichiers et les différents processus mis en place n'ont pas été organisés comme une chaîne de traitement globale. Ainsi, le premier travail a été de hiérarchiser les fichiers, afin de définir une base stable pour les futures exécutions.

I.2.1 Une nouvelle arborescence des dossiers

Depuis 2016, la méthode utilisée pour modifier ou ajouter des parties de la chaîne de traitement était simple : chaque étudiant créait un répertoire, dans lequel il mettait d'un côté les travaux précédents, et de l'autre les différents dossiers nécessaires au bon fonctionnement des algorithmes qu'il développait.

L'ensemble de cette organisation a été revue, comme détaillée dans la Figure 2.

Figure 2 : Nouvelle arborescence des dossiers de la chaîne de traitement



Cette organisation a été pensée notamment pour la gestion des données. Ainsi, une personne ayant peu de connaissances dans la vectorisation d'image pourra très facilement lancer le programme, en mettant les données initiales dans le dossier « donnees_init », et à la fin, il pourra voir les résultats qu'il attend dans le dossier « donnees_fin ».

Pour avoir plus de détails, il faut directement aller dans le dossier « resultats » de l'étape correspondante. Cette nouvelle arborescence implique aussi une révision totale du programme.

I.2.2 Le squelette du programme

Deux objectifs ont présidé à la révision du programme :

- Pouvoir lancer la chaîne de traitement en une fois, et la laisser se dérouler sans qu'il n'y ait d'interruption, sauf cas prévu dans le processus.
- Ajouter un mode test, permettant de pouvoir essayer de nouvelles fonctionnalités sur plusieurs données à la fois, sans que cela n'empêche une utilisation du programme en mode normal.

Pour le premier point, il a suffi d'organiser les programmes selon l'arborescence précédente, en faisant attention aux différentes entrées et sorties demandées pour chaque morceau de programme. L'organisation du nouveau programme est présentée en Figure 3.

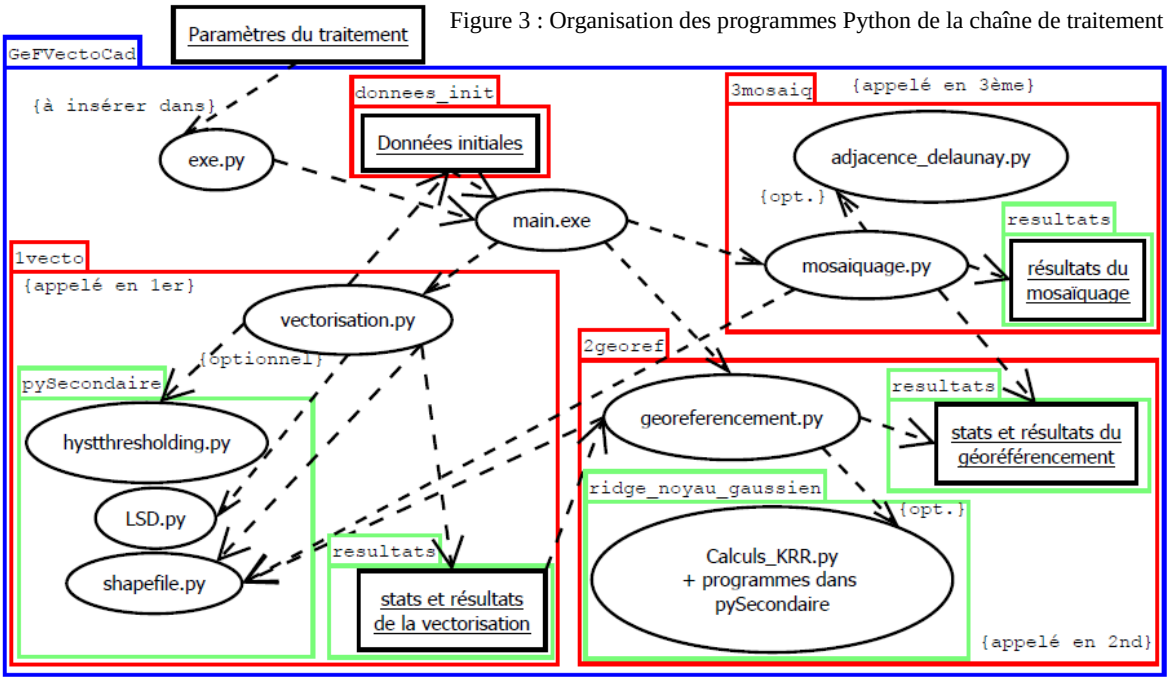


Figure 3 : Organisation des programmes Python de la chaîne de traitement

Le second objectif est important, notamment si le programme tourne longtemps. Dans la version précédente, les paramètres des différentes étapes n'étaient pas tous gérés depuis un emplacement unique. Il faudrait utiliser un fichier de type txt pour réunir ces paramètres. Mais cette méthode présente trois inconvénients :

- Une seule exécution est possible (ou alors, il faudrait écrire plusieurs fichiers txt pour chaque exécution, mais lorsque on ne veut pas exécuter l'un des fichiers, il faut le supprimer du dossier) ;
- Il faut suivre une certaine mise en forme, pour que Python puisse lire correctement les paramètres ;
- L'interprétation de certains types de variables nécessite « décodeur » pour comprendre à quoi ça correspond. Par exemple, parmi les paramètres, il faut spécifier la fonction permettant de transformer en niveau de gris une image ; il faut alors faire comprendre à Python que la chaîne de texte correspond à une fonction particulière, ce qui, s'il y a un nombre important de fonctions, peut devenir lourd à programmer.

Pour résoudre tous ces points, une alternative a été trouvée, avec un fichier exe.py.

```
# -*- coding: utf8 -*-
#!/home/anthony/miniconda3/bin/python3.7
from main import *

main(commune                = 'Aubigne-Racan',
     annee                  = 1850,
     planches               = ['114','115'],
     type_images            = 'JPG',
     prefixe                = 'N50-1008060',
     #reduction              = [[2000,3000],[2000,3000]],
     fct_NG                 = rgb2gray,
     max_niveau_print       = 3,
     couleurs_ligne         = [255,0,0],
     # Paramètres vectorisation
     ## Pré-traitement (des 2 images)
     v_i1_chemin_masque     = ['114_masque.tif','115_masque.tif'],
     v_i1_inverse           = True,
     v_i1_binaire           = False,
     v_i1_simpl             = ('hysteresis_local',1,(1.10,None),'maximum'),
     v_i2_chemin_masque     = ['114_masque_carroyage.tif','115_masque_carroyage.tif'],
     v_i2_inverse           = True,
     v_i2_binaire           = True,
     v_i2_simpl             = ('hysteresis_local',5,(1.10,1.20),'binaire'),
     v_rso_taille_objet     = 100,
     ## Détection (avec image 1)
     v_meth                 = 'line_segment_detector',
     v_lsd_decoup_axe       = 4,
     #v_lsd_tau              = np.pi/8,
     ## Post-traitement (avec image 2)
     #v_min_intensite        = 1.0,
     v_min_pourcent         = ((100,0.80),0.60),
     v_epaisseur            = 2.5,
     v_max_long_erreur      = 20,
     v_prec_finale          = 1,
     v_correct_auto         = True,
     v_chemin_compar_lim    = ['114_limites_ref.tif','115_limites_ref.tif'],
     v_chemin_compar_parc   = ['114_parcelles_ref.shp','115_parcelles_ref.shp'],
     # Paramètres autres étapes
     g_meth                 = "Régression ridge à noyau gaussien",
     m_meth                 = False,
     m_graphe_adjacence     = True,
     m_distance_densif      = 10,
     #chemin_granulaire      = None,
     #chemin_temporel        = None,
     )
```

Figure 4 : Capture d'un morceau du fichier exe.py, ouvert avec l'IDLE de Python

Comme nous pouvons le voir, la mise en page est claire et personnalisable. On peut, par exemple, insérer des commentaires précédés d'un dièse (comme la ligne 13), désactiver une option (comme pour la ligne concernant le paramètre « reduction ») et on peut ajouter autant d'exécutions à la suite (en pensant à modifier le paramètre « prefixe » pour différencier les processus), qui s'enchaîneront dans l'ordre indiqué par le fichier exe.py. La première ligne permet d'appeler le fichier main.py. Puis, nous appelons la fonction main (incluse dans main.py) qui lance l'ensemble de la chaîne de traitement, avec à chaque ligne un paramètre personnalisable. Un rapide mode d'emploi (avec le détail sur les paramètres disponibles) est proposé en Annexe 4.

Cette première étape a largement facilité les ajouts dans la chaîne de traitement. C'est grâce à cela que nous pouvons présenter des résultats concrets sur une donnée après l'exécution de la chaîne complète (voir le paragraphe IV.2.3)

II Nouvelles contributions à l'étape de vectorisation

II.1 Etat de l'art

L'automatisation de la vectorisation des planches cadastrales font l'objet d'une multitude de travaux de recherches. En France, c'est dans les années 1990 que la DGI (aujourd'hui DGFiP), sous l'impulsion des collectivités publiques, commence à numériser et dessiner manuellement les parcelles (1). Au 1er avril 2019, 33 682 communes sont couvertes par le PCI Vecteur sur un total de près de 36 000[3]. Mais aujourd'hui, l'automatisation devient plus que nécessaire pour créer un historique du parcellaire.

II.1.1 Des travaux de recherches pour améliorer la vectorisation des images

Lors des recherches de son TFE en 2017 (2), Charlotte Odie avait déjà référencé une grande partie des travaux de recherches en cours sur la vectorisation (paragraphe 1.2.2 de son mémoire).

En complément, nous pouvons citer les travaux de Mauricio Giraldo Arteaga datant de 2013 (7). Le *New York Public Library Labs* possède un ensemble de planches en couleurs, mettant notamment en avant la disposition des bâtiments sur les terrains, et un ensemble d'informations permettant de désigner individuellement chacune des constructions. M.G. Arteaga a élaboré un processus semi-automatique permettant la vectorisation des bâtiments. La chaîne est divisée en quatre principales étapes : un pré-traitement manuel, la détection des polygones, la simplification des arêtes et la suppression des polygones inutiles. Cette chaîne est impossible à appliquer aux planches du cadastre français dont la qualité est bien moindre. Les planches états-uniennes possèdent une légende bien précise, multicolore, et sans ambiguïté, et les textes, de très bonne qualité, sont associés à une seule entité géométrique. Néanmoins, certaines actions sur les données sont intéressantes pour notre étude. Par exemple, la détection et la simplification des polygones se réalise en croisant un quadrillage avec la frontière de la zone caractérisée par la même couleur (voir Figure 5). Les bords extérieurs du quadrillage ainsi déterminés correspondent au polygone, avec un nombre

[3] Source des données chiffrées : www.data.gouv.fr (5). Concernant les communes non vectorisées, 88% se situe dans le Nord-Est de la France, et en particulier dans l'ex Champagne-Ardennes où la moitié des communes de cette ancienne région ne sont encore pas vectorisées (6).

de sommets simplifiés. Ce processus, qui dépend de la couleur, est toutefois difficile à appliquer au cadastre français, où le quadrillage devra s'arrêter aux limites des parcelles, ce qui est plus complexe à mettre en œuvre.

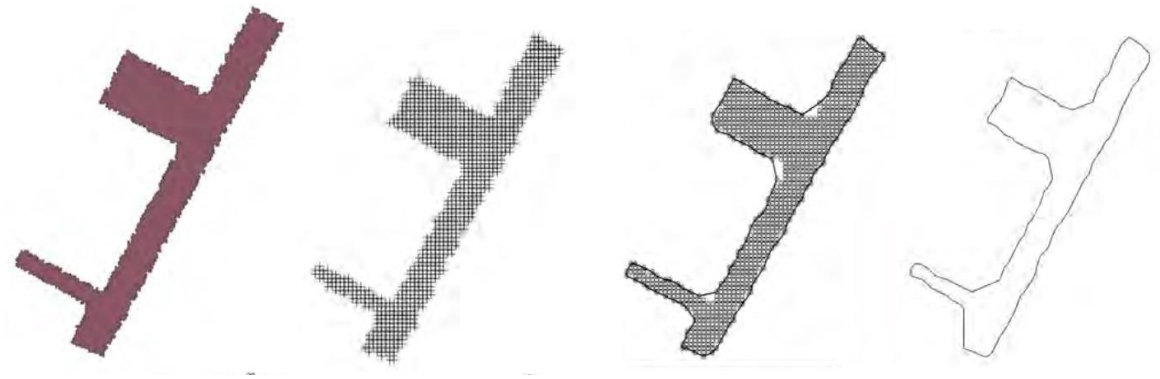


Figure 5 : Illustration des étapes du processus de détection d'un polygone (M.G. Arteaga, 2013)

Les travaux de R. Cura, B. Dumenieu, N. Abadie, B. Costes, J. Perret et M. Gribaudi en 2017 (8) diffèrent de la vectorisation directe. Le principe est de géoréférencer des données géographiques écrites (parfois non situées sur un plan), grâce à des données existantes (comme par exemple des données foncières type droit de propriété, qui ne donne qu'une position relative par rapport à d'autres bâtiments, sans aucune carte jointe). Le processus qu'ils ont mis en place consiste à trouver la meilleure correspondance entre une donnée initiale et une donnée stockée dans une base plus ou moins récente (selon la disponibilité de cette donnée). Leurs travaux traitent de plusieurs problématiques comme le souci du type d'information qui est traité, l'obtention d'un taux de validation selon la qualité des solutions trouvées… Comme l'une des finalités est de réaliser cette vectorisation par un travail collaboratif (le *volunteered geographical information*), cette opération ne peut donc servir à la vectorisation directe et semi-automatique dans notre chaîne de traitement. Certaines idées sont tout de même exploitables, en particulier la détection d'informations autres que les limites de parcelles (caractères, symboles), améliorable en utilisant du *deep-learning*.

Cet inventaire de travaux de recherches n'est pas exhaustif, car ce sujet est au cœur des priorités des services qui s'occupent de réaliser des bases multi-datées d'information géographique. Ainsi, le laboratoire a voulu développer sa propre chaîne de traitement et proposer un autre processus.

II.1.2 Les applications à la chaîne de traitement

Dans la chaîne de traitement antérieure, la vectorisation nécessite deux étapes essentielles : la détection des segments et la labellisation. La vectorisation a une forte influence sur le reste du processus, car c'est la première étape de la chaîne de traitement. À l'issue de celle-ci, la donnée de base du traitement (la planche) ne sera plus utilisée, donc aucun autre contrôle qualité ne pourra être réalisé. Deux processus de détection de segments ont été testés dans la chaîne de traitement du laboratoire (2) :

- La transformée de Hough probabiliste (HP), via le module Python « skimage » (9). C'est un algorithme rapide qui utilise les probabilités pour détecter les lignes. Néanmoins, pour des images initiales très bruitées comme nos données, cet algorithme peut s'avérer insuffisant du point de vue qualitatif (présence de segments multiples, cf. II.2.2)
- L'algorithme de vectorisation du logiciel GRASS *r.to.vect*. Cet algorithme donne des pourcentages de détection équivalents à la précédente méthode. La principale différence se trouve dans sa mise en œuvre, puisque l'algorithme n'a pas besoin de paramètres initiaux.

D'après les conclusions de Charlotte Odie, sur trois planches de la commune d'Aubigné-Racan, la vectorisation était plutôt correcte pour les années 1850 et 1972 (entre 92% et 100% de bonnes détections des parcelles et entre 0% et 8% de fausses alarmes), mais mauvais pour l'année 1813 (entre 10% et 11% de bonnes détections de parcelles). En réalité, une erreur de programmation sur le calcul de ces statistiques fausse les chiffres obtenus. Finalement, ses résultats sont meilleurs qu'annoncés avec notamment 92% de bonnes détections pour 1813. Mais la qualité de la vectorisation est toujours le gros point faible de ces deux méthodes.

Jean-Marc Beveraggi a ensuite émis plusieurs remarques sur la réalisation de cette étape au début de son TFE (3), remettant en cause l'efficacité de ces processus. Il était donc nécessaire de trouver une nouvelle méthode de vectorisation, comme le *Line Segment Detector*, qui semble prometteur (4). Dans la suite, lorsque nous ferons référence aux « méthodes appliquées précédemment », il s'agira des méthodes utilisées dans la chaîne de traitement décrite ci-dessus, c'est-à-dire HP et la vectorisation par GRASS.

II.2 L'algorithme à implémenter : le *Line Segment Detector*

II.2.1 Présentation synthétique de l'algorithme

Cette méthode de détection de ligne a été développée par quatre chercheurs : Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel et Gregory Randall en 2012 (4). Elle permet de détecter n'importe quelle séparation présente sur une image (limite, frontière, arête…), sans passer au préalable par une détection de points de contours. L'algorithme étudie les discontinuités de niveaux de gris plus ou moins brusques pour reconnaître ou non une ligne de contour potentielle. Un certain nombre de contrôles permet ensuite de supprimer la plupart des artefacts.

Ce procédé répondrait à notre problématique, à savoir améliorer la détection des limites de parcelles. C'est pourquoi, nous avons évalué son intérêt dans le cadre de notre chaîne de traitement.

Avant de détailler les avantages et inconvénients de l'algorithme, nous avons rencontré des limites dans la précision des variables flottantes et les limites de calcul dans Python. C'est particulièrement le cas pour la détermination du NFA (Number of False Alarms) qui est défini dans l'article (4). C'est un critère de validation des rectangles associés à chaque segment détecté ; sa valeur doit être inférieure à 1, et plus elle est proche de 0, plus le rectangle est précis. Ce critère est souvent calculé, et nous obtenons des exponentielles de nombres très petits[4]. Ainsi, dans les points précédents, lorsque nous devons évaluer des valeurs de NFA, en réalité nous comparons des logarithmes népériens de ces valeurs.

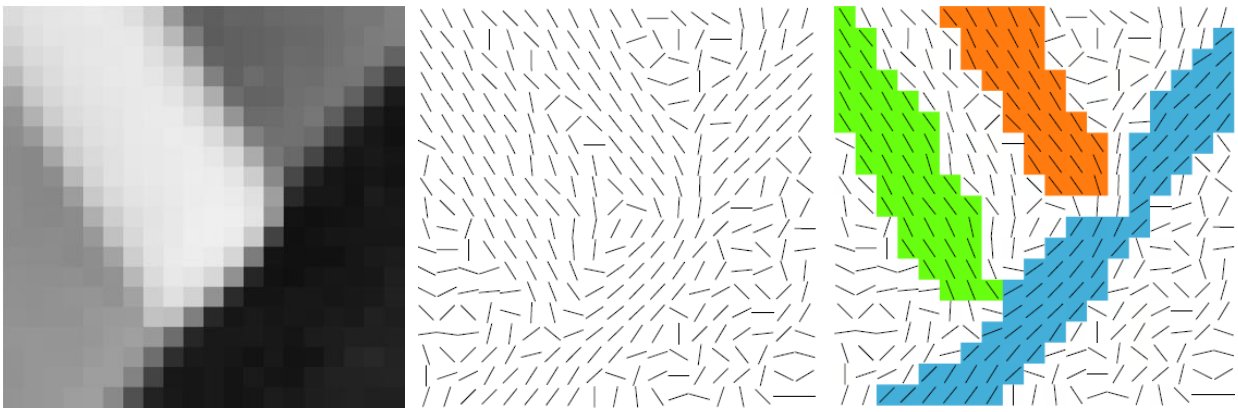


Figure 6 : Exemple de la détection de trois segments (vert, orange et bleu), résumant LSD (4)

[4] Le NFA utilise la fonction Gamma, mais pour accélérer le calcul, une formule approchée est employée (10).

II.2.2 Avantages et inconvénients

LSD possède certaines propriétés, plus ou moins avantageuses, par rapport aux méthodes utilisées précédemment ;

1° Robustesse :

Bien qu'il ait besoin de six paramètres initiaux, différentes preuves, apportées par les chercheurs eux-mêmes (4) et au vu des résultats recensés sur un site internet (un utilisateur quelconque peut réaliser une détection grâce à la plateforme en ligne), permettent de garder une valeur unique à chaque paramètre sans se préoccuper de l'image à vectoriser. Néanmoins, dans notre cas, un des six paramètres va être testé : la tolérance liée à l'orientation du gradient (explications dans le paragraphe IV.2.1) ;

2° Complexité de l'algorithme :

LSD est un algorithme très gourmand, car il visite les pixels individuellement et, à chaque pixel visité, il étudie tous les autres pixels voisins non utilisés. Le temps d'exécution peut donc exploser. En toute logique, le nombre de tests à réaliser entre les pixels diminue avec la taille de l'image. Ainsi, nous avons eu l'idée de réaliser le LSD en plusieurs fois, en découpant l'image. Des éléments de comparaison sont détaillés dans le paragraphe IV.2.1 ;

3° Propreté des résultats :

Contrairement aux méthodes précédentes (voir Figure 7, image a), LSD permet de ne détecter que deux segments au maximum sur un morceau d'une limite (un segment à chaque bord de la limite – voir Figure 7, image b). Ainsi, le post-traitement est beaucoup plus rapide, et la vectorisation des parcelles est beaucoup plus simple à réaliser. De plus, l'algorithme est insensible au bruit ;

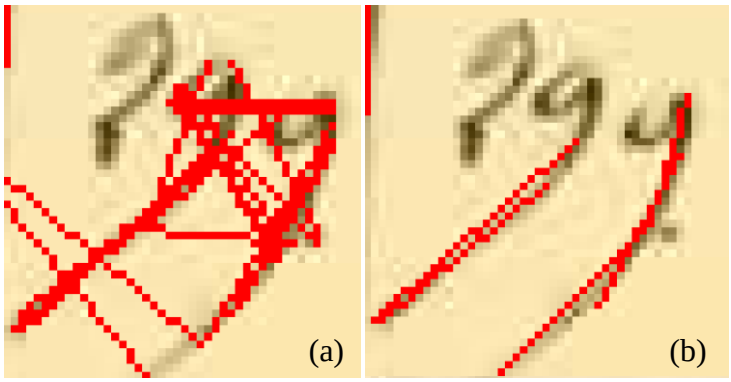


Figure 7 : Exemple de lignes détectées avec HP (a) et LSD (b)

4° Application à une image non binarisée :

LSD a besoin d'une image non binarisée pour fonctionner correctement. C'est à la fois un avantage et un inconvénient. En effet, cela permet d'avoir chaque pixel avec un poids différent. Mais, sur une image simple (par exemple, une image binarisée initialement), une des méthodes précédentes parait suffisante pour la détection de lignes ;

5° Détection des lignes

LSD est pensé pour tracer des contours de type marche en escalier ou rampe. Son objectif est donc différent de celui que nous recherchons : nous voulons détecter les traits foncés tracés entre des zones plus claires présentant des niveaux de gris similaires (contours de type toit ou pic). C'est le problème majeurauquel nous avons dû faire face, car dans la chaîne de traitement, le décalage du segment détecté par rapport au trait existant pose un problème dans l'étape de validation. La solution proposée se trouve dans le paragraphe II.3.2.

En résumé, l'algorithme présente plus d'avantages que d'inconvénients pour notre approche. Néanmoins, il est difficilement adaptable, à la vue des points 1 et 5. C'est pourquoi, nous avons décidé de l'implémenter nous-mêmes en Python et de modifier le prétraitement et le post-traitements réalisés les années précédentes (avec l'algorithme HP) pour obtenir les meilleurs résultats possibles.

II.3 Les modifications du programme initial

Initialement, l'ensemble de l'étape de vectorisation a été programmé par Charlotte Odie, lors de son TFE de 2017 (2). Quelques modifications avaient été apportées ensuite par ses maîtres de stage, mais aussi par Jean-Marc Beverraggi lors de son TFE de 2018 (3). Néanmoins, le programme développé ne permettait pas de modifier des paramètres ou de personnaliser l'exécution de la chaîne. L'un des objectifs a donc été de réaliser un nouveau programme, qui permet de choisir la méthode de vectorisation employée, grâce à la personnalisation des paramètres.

Pour plus de clarté, nous avons privilégié une méthode de détection selon l'année de la planche vectorisée. Les résultats dans la partie IV.2 expliquent la raison de ce choix.

II.3.1 Ajout de fonctionnalités au prétraitement

À l'origine, le pré-traitement dépendait de l'année des planches du cadastre ; l'ordre des étapes du pré-traitement est rappelé dans la Figure 8 (le masque contour correspond au masque pour éliminer les écritures en dehors des parcelles). Cette succession d'opérations donnait de bons résultats lorsqu'elle s'appliquait aux précédentes méthodes de vectorisation. Mais après quelques tests avec LSD, le pré-traitement de départ n'était plus adapté.

Première observation : LSD a besoin d'une image non binarisée, car il classe des pixels selon la norme de son gradient. Donc, toutes les étapes utilisant ou sortant une image binaire (la binarisation, la squelettisation et la suppression des objets isolés) sont supprimées.

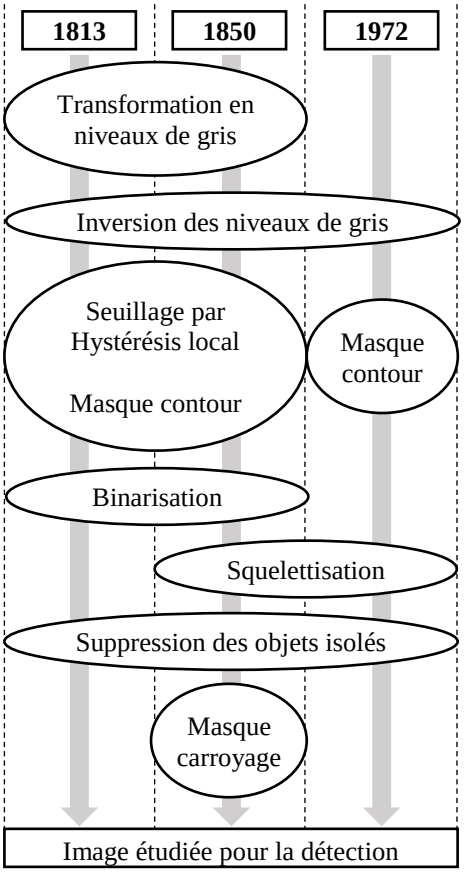


Figure 8 : Étapes du pré-traitement pour la vectorisation, réalisées par Charlotte Odie, avec l'exemple des planches d'Aubigné-Racan

Du pré-traitement tel qu'il était initialement réalisé, nous avons conservé principalement le seuillage par hystérésis local, avec quelques modifications pour LSD. Au préalable dans ce seuillage, après réduction du niveau de gris à l'intervalle [0,1], nous calculions une moyenne des niveaux de gris de chaque pixel et ses voisins (voisinage 3x3 pour la détection et 11x11 pour la validation), pour déterminer un seuil bas et un seuil haut

(différent pour chaque pixel) ; si le niveau de gris d'un pixel était supérieur au seuil haut, alors la valeur du pixel changeait pour 1 ; si le niveau de gris d'un pixel était inférieur au seuil bas, la valeur du pixel changeait pour 0 ; sinon, la valeur du pixel changeait pour 1 (respectivement 0) si le pixel était (respectivement n'était pas) connecté à un pixel qui avait pour valeur 1. Ainsi, le résultat se présentait sous forme d'une image binaire. Le traitement a été adapté en ne gardant qu'un seuil bas local (le seuil haut n'étant plus pris en compte) et en appliquant un maximum local pour les pixels qui aurait dû recevoir la valeur 1 (les pixels devant recevoir 0 gardent cette valeur). Ceci permet d'accentuer les lignes à détecter et, au contraire, d'atténuer les pixels de bruit. Pour cette adaptation, il est nécessaire d'inverser l'image avant le seuillage car LSD détecte plus facilement des lignes à proximité de pixels dont le niveau de gris est élevé (blanc).

Une autre modification a été apportée aux étapes basiques d'inversion (et de binarisation et de suppression des objets isolés si LSD n'est pas utilisé). Le seuillage et la squelettisation ont besoin de l'inversion et/ou de la binarisation pour fonctionner correctement. Mais l'application du masque, effectuée avant la squelettisation ou après le seuillage, nécessite une image non inversée. Nous sommes donc contraints de réaliser ces étapes basiques à la fin du prétraitement. Si besoin, ces étapes sont intégrées au procédé principal (c'est le cas pour la squelettisation et le seuillage), et l'opération opposée sera effectuée juste après. La Figure 9 résume l'ensemble du pré-traitement.

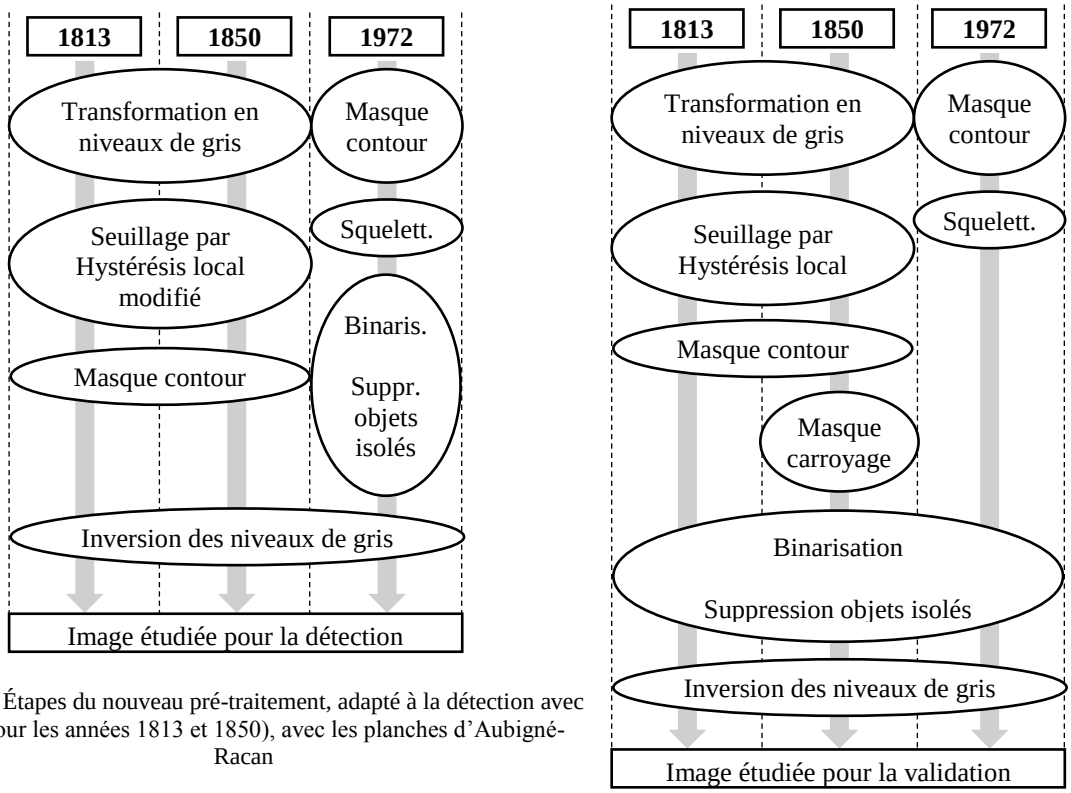


Figure 9 : Étapes du nouveau pré-traitement, adapté à la détection avec LSD (pour les années 1813 et 1850), avec les planches d'Aubigné-Racan

Deux points à souligner :

- Le processus diffère pour le cadastre de 1972, qui utilise HP et non LSD pour la détection des segments (voir les parties IV.2.2 et IV.2.3). C'est pourquoi, nous retrouvons les étapes de binarisation, de squelettisation et de suppression d'objets isolés.
- La différenciation entre l'image servant à la détection et l'image servant à la validation, nécessaire pour la validation en post-traitement des segments, grâce à l'utilisation d'une image binaire.

II.3.2 Modification du LSD pour le cadastre

Comme il a été dit précédemment, LSD est un algorithme difficile à modifier, sans altérer lourdement la détection. La modification du pré-traitement est donc requise. Par ailleurs, l'utilisateur doit pouvoir avoir le choix entre plusieurs algorithmes de détection ; dans notre cas, LSD ou HP. Il faut donc que l'entrée et la sortie soient dans le même format.

L'insertion de l'image seuillée dans LSD suffit à obtenir un résultat correct. Pas besoin donc de modifier le traitement. Mais, comme cela est expliqué dans le point 5 du paragraphe II.2.2, LSD détecte deux segments par frontières à détecter, un de chaque côté. Cela pose un souci pour le post-traitement. Pour y pallier, deux techniques ont été testées :

- La modification initiale des gradients. Elle consistait à modifier la norme des gradients pour privilégier le centre des lignes. Ainsi, à chaque pixel, après le calcul des gradients, l'algorithme étudiait les gradients voisins. Si le gradient le plus fort était celui du pixel en étude, alors la norme était remplacée par la norme minimale locale. Si ce n'était pas le cas, le gradient était modifié par la valeur la plus forte. L'angle choisi ensuite correspondait à ce maximum.

  Cette méthode fonctionnait remarquablement bien sur plus de la moitié des segments. Malheureusement, beaucoup n'étaient pas détectés ou doublés, quand les angles étaient proches de valeurs remarquables (0, 90°, 180°…) ou lorsque la norme du gradient variait de façon aléatoire à cause de la qualité de la planche. C'est pourquoi, cette méthode n'a pas été retenue.

- Le décalage des segments détectés. Cette méthode est très intuitive puisqu'elle consiste à décaler les segments vers le centre de la limite. Les

limites de parcelles ayant une épaisseur plutôt homogène sur l'ensemble de l'image, le décalage des segments se réalise parfaitement. Néanmoins, il appartient à l'utilisateur de fixer ce dernier paramètre. Malgré cela, cette méthode a été retenue.

Cette deuxième option correspond aux modifications apportées à la fin du LSD. Elle n'est pas associée au post-traitement car elle ne concerne pas LSD.

II.3.3 Révision du post-traitement

Les résultats sortant des algorithmes de détection sont difficilement exploitables. Avec HP, les segments sont démultipliés, et avec LSD, les segments ne s'intersectent pas. De plus, plusieurs frontières ne sont pas détectées, soit parce qu'elles sont plus effacées, soit parce qu'elles sont très proches les unes des autres.

Le TFE de Charlotte Odie apportait un premier post-traitement, qui se déroulait ainsi :

- Un découpage simple des segments aux intersections (l'ensemble des intersections n'était cependant pas traité) ;
- Une validation des sous-segments en étudiant l'image binaire ;
- La création de jointures simples entre extrémités existantes de segments.

Ce post-traitement était assez rapide, mais peu efficace, car il ne traitait pas l'ensemble des problèmes causés par la détection. Un nouveau post-traitement a donc été créé, qui utilise quatre fonctions principales ;

- *sous_segments* : découpe des segments sécants, puis valide chaque sous-segment en vérifiant la présence de points de contours sur l'image de validation. Un taux de validation (pourcentage de pixels du segment rasterisé correspondant à des points de contour) est alors associé à chaque segment ;
- *fusions* : rassemble en un seul segment les segments parallèles (écartés d'une fois l'épaisseur) ou les segments qui se poursuivent et qui ont la même orientation (dont au moins une extrémité est suffisamment proche entre les deux segments). Pour cela, une fonction annexe a été développée (*segments_paralleles*). Elle consiste à vérifier le parallélisme d'une droite, en introduisant une précision (nous avons appelé ces droites, des droites

« pseudo-parallèles selon une précision p », où p représente le paramètre d'épaisseur – cf. Annexe 5) ;

- *prolongements* : prolonge des segments dont l'intersection est possible sur la planche, et dont la validation indique l'utilité de ce prolongement. Un seul test de prolongement non valide est permis par extrémités, pour limiter au maximum la détection de segments dans des espaces non-cadastrés type route (cas le plus fréquent de fausse détection après un mauvais prolongement). Cas B de la Figure 10 ;
- *jointures* : ajoute des segments qui permettent de rejoindre deux extrémités de segments. Chaque jointure est alors validée ou non avec les limites tracées sur l'image servant à la validation. Cas C de la Figure 10.

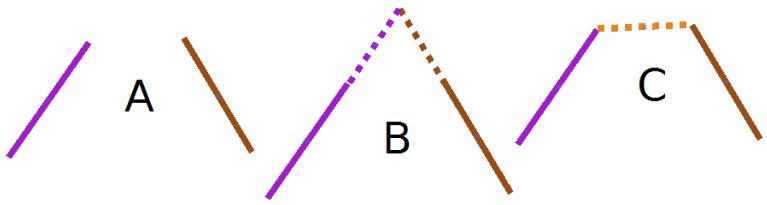


Figure 10 : Illustration pour différencier *prolongements* et *jointures*. En A, l'état initial, en B, les morceaux de segments créés avec *prolongements*, et en C le nouveau segment créé avec *jointures.*

Différents tests ont été réalisés, en essayant de garder le moins d'étapes possibles. Il existe deux types de fonctions :

- Les fonctions de créations de segments, pour combler des manques aux détections. Ce sont *prolongements* et *jointures*. Ces fonctions sont nécessairement précédées et suivies des fonctions de correction ;
- Les fonctions de correction, qui suppriment ou corrigent des segments a priori faux, fusionnent des intersections ou suppriment et découpent des segments. Ce sont les fonctions *sous_segments* et *fusions*.

Cette distinction est importante, car elle permet de mieux comprendre leur utilité. D'une part, les fonctions de création de segments sont logiquement exécutées une seule fois dans le processus. D'autre part, les fonctions de correction, elles, s'exécutent à chaque résultat obtenu, car elles permettent de corriger les erreurs.

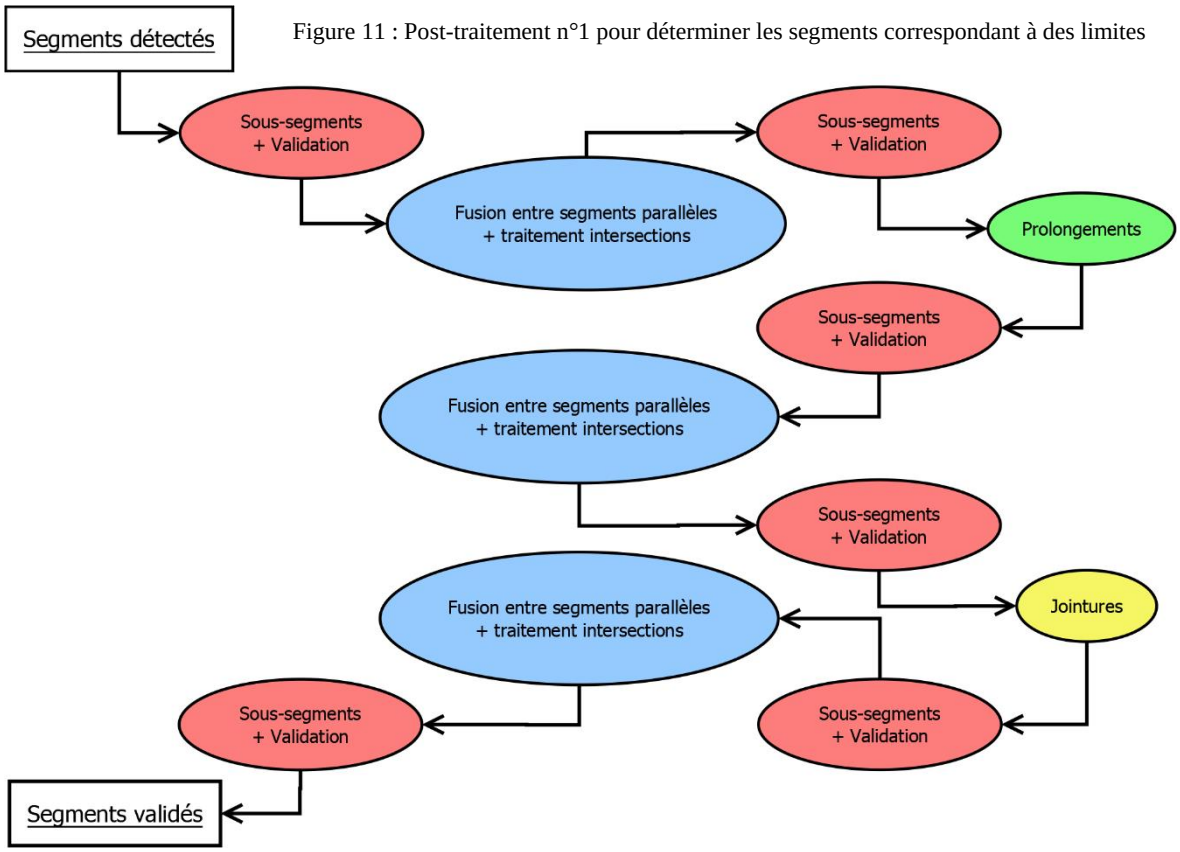


Figure 11 : Post-traitement n°1 pour déterminer les segments correspondant à des limites

La Figure 11 est le résumé du premier post-traitement, qui permet d'obtenir un ensemble de segments valides et simplifiés. Néanmoins, certaines connexions entre des segments manquent encore et entraînent la formation d'énormes polygones qui englobent plusieurs parcelles. Un second post-traitement est donc nécessaire.

Nous partons du principe suivant : les segments validés à la fin du premier post-traitement sont « parfaits », c'est-à-dire qu'ils correspondent obligatoirement à une limite sur la planche. Cette affirmation est fausse, car nous procédons à des validations. Mais elle est suffisamment proche de la réalité, et nécessaire pour utiliser d'autres traitements.

Le post-traitement qui suit s'appuie donc sur ce postulat, mais, les étapes proposées entre la lecture des segments valides à la première formation des polygones ne sont pas figées, car elles dépendent de la qualité des détections. Ici, nous proposons une solution pour traiter le cas où seules des frontières ont été détectées et où l'on ne dispose pas de données exogènes. L'ajout de la détection des caractères, la lecture du tableau d'assemblage ou encore le pointage manuel d'un point dans chaque parcelle, simplifieront cette partie du second post-traitement, le reste étant nécessaire (voir le paragraphe III.2.2).

Ce nouveau traitement est découpé en deux parties.

La première partie permet de simplifier davantage les segments et de remplir les trous qui existerait entre deux segments qui auraient dû se rejoindre. On réutilise notamment les fonctions développées précédemment, mais sans réaliser de validation avec la planche initiale. Nous commençons par fusionner les segments avec *fusions*, avec une tolérance plus grande (1,5 fois l'épaisseur au lieu de 1 fois précédemment – voir Figure 12) ; puis, nous découpons en sous-segments avec *sous_segments*, en contrôlant les intersections (fusions des points éloignés de moins de 1,5 fois le paramètre d'épaisseur) ; nous supprimons les petits segments formant des noeuds pendants (segments dont l'une des extrémités ne rejoint pas d'autres segments) qui empêcheraient de réaliser des jointures correctes (voir Figure 13) ; nous supprimons les segments dont les deux extrémités sont complètement isolés ; nous tentons de rejoindre tous les segments pendants grâce à *jointures* ; enfin, nous redécoupons les segments avec *sous_segments*, en contrôlant les intersections (uniquement si les points sont des extrémités et sont éloignés de moins de 1,5 fois le paramètre d'épaisseur).

Figure 12 : Exemple de fusion nécessaire pour simplifier les segments en double

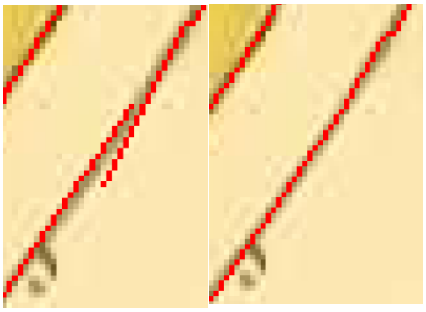


Figure 13 : Exemple de suppression d'un noeud pendant, permettant de créer une jointure correcte

À l'issue de cette première partie, des polygones sont détectés en utilisant la fonction *polygonize_full* de Shapely[5]. Quelques segments ne sont pas traités et sont renvoyés tels quels ce qui nécessite une seconde partie pour les traiter.

La fonction *polygonize_full* de Shapely (11) renvoie trois types de segments : les segments invalides (en particulier ceux qui se croisent alors qu'ils n'ont pas de sommets communs), les segments isolés (ceux-ci sont pour le moment abandonnés) et les segments qui devraient être des côtés de polygones, mais qui finalement ne le sont pas. Ce dernier cas

[5] La fonction *polygonize_full* du module Shapely est une fonction permettant de créer un ou plusieurs objets de types Polygon (objets de Shapely) grâce à un ensemble de groupe de points formant des lignes, qui doivent nécessairement s'intersecter sur des points faisant partie de ces groupes de points. La fonction retourne des polygones, mais aussi des lignes. (11)

est problématique car il semblerait que ce soit la fonction elle-même qui soit en cause (absence de logique apparente, avec des segments qui auraient dû être intégrés dans un polygone, et qui se retrouvent ainsi dans cette catégorie). Il est donc impossible d'y remédier sans un traitement ultérieur. Trois étapes sont nécessaires pour traiter ces erreurs.

- La création des polygones manquants quand les limites sont contenues dans les polygones (devant former un autre polygone – voir Figure 14). Le processus va étudier les segments contenus dans les polygones. Il crée des buffers autour de ces segments, et il détermine la différence entre l'ensemble des polygones et les buffers des lignes. Le trou formé par le buffer de la ligne qui divisait est comblé en modifiant les coordonnées des nouveaux polygones selon les coordonnées de celle-ci.

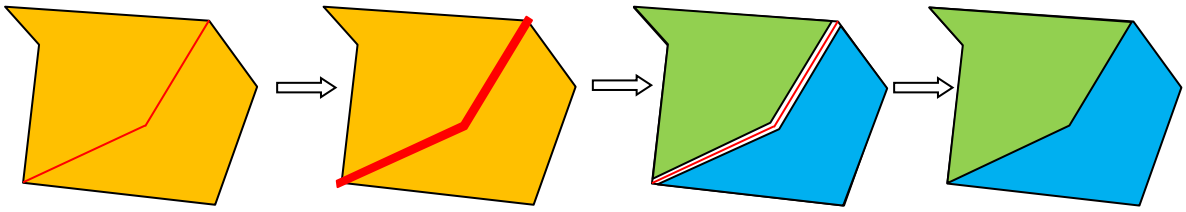


Figure 14 : Processus de fabrication d'une division de polygones

Ce processus complexe compense les limites du module Shapely (11), qui ne permet pas de découper directement des polygones.

- La création des polygones manquants correspondant à un trou (à l'intérieur ou bord de l'ensemble des parcelles – voir Figure 15). Sur le même principe que dans le paragraphe précédent, nous créons un buffer des lignes qui n'appartiennent pas à des polygones, en excluant les nœuds pendants. Puis, nous prenons l'enveloppe convexe de l'ensemble des parcelles, et nous faisons la différence avec la fusion de l'ensemble des polygones et des buffers des lignes. De nouveaux polygones sont créés autour de ces zones en considérant les extrémités des segments préexistants. Cette méthode ignore les « encoches » (zone A) car l'un des bords n'existe pas, et ne garde que les « vraies » parcelles (telles que la zone B).

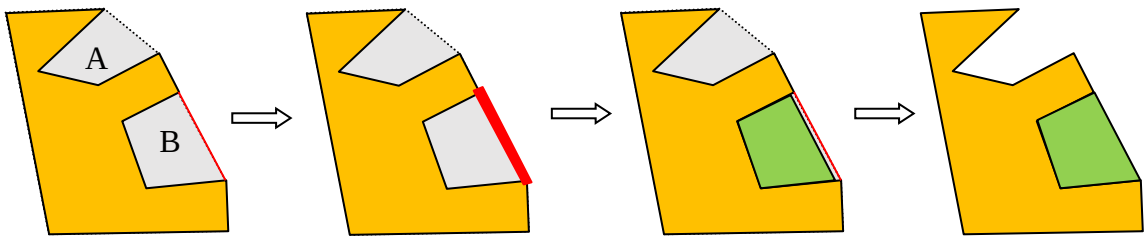


Figure 15 : Processus de création de polygones dans un trou (en gris, l'enveloppe convexe virtuelle des parcelles)

❖ La fusion des petits polygones (dont la superficie est inférieure à 40 fois - valeur arbitraire - le produit de l'épaisseur du trait sur l'image (paramètre initial) par la longueur minimale d'un segment fixée au début). Pour cela, le programme étudie l'intersection entre le petit polygone et l'enveloppe convexe des parcelles alentours. Le polygone dont l'enveloppe convexe recouvre le plus le petit polygone est choisi pour absorber la petite parcelle.

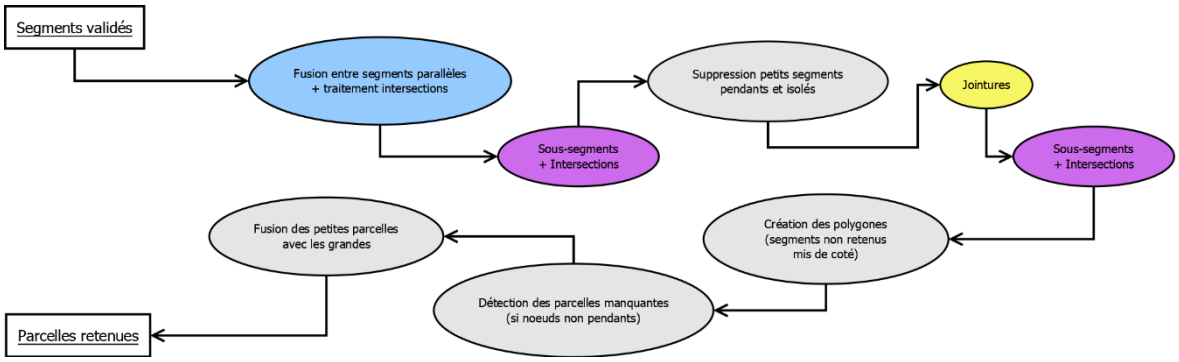


Figure 16 : Schéma du second post-traitement

À l'issue de ce second post-traitement (résumé dans la Figure 16), de nouveaux polygones sont ajoutés à la liste. À ce niveau de la chaîne de traitement, il reste quelques erreurs topologiques. C'est donc à cette étape que l'opérateur doit pouvoir modifier le contenu. Pour cela, il aura à sa disposition la planche de départ, les parcelles détectées, et les segments validés (avec le taux de validation) à l'issue du post-traitement. L'idée est de guider au mieux l'opérateur dans sa démarche pour lui faciliter le travail. Avec ces informations, et après ce post-traitement, la correction devrait se faire très rapidement.

III La théorie des graphes, un outil essentiel à l'amélioration de la chaîne de traitement

III.1 Etat de l'art

III.1.1 Les travaux de recherches

La théorie des graphes est un outil mathématique puissant qui permet, dans notre cas, d'établir des relations entre des entités spatiales, grâce à la topologie ou la hiérarchie par exemple[6]. Née il y a presque trois siècles, elle est toujours le sujet de plusieurs recherches.

Énormément de données historiques associées à des situations géographiques ne sont pas traitées à cause de la difficulté d'élaboration de base de données numériques et géographiques. Beaucoup de travaux de recherches utilisent les graphes pour les exploiter. C'est le cas de Rodier et al. en 2014 (13), dont l'objectif est de pouvoir relier dans un modèle unique, des données hétérogènes (registres, manuscrits) à des plans en combinant un maximum de détails de positionnement (absolu ou relatif). Ces travaux ont porté sur la paroisse d'Odars (au sud-est de Toulouse), en utilisant de nombreux documents fonciers remontant aux derniers siècles du Moyen-Age (plans terriers, compoix méridionaux). Dans notre cas, nous n'utilisons que des plans qui portent jusqu'à présent sur des zones rurales, ce qui est très différent, notamment concernant la disponibilité des données. Mais, la chaîne pourrait très bien être complétée par des informations provenant des documents d'arpentage.

Il existe aussi beaucoup de travaux détaillant l'utilité de chaque graphe. On peut par exemple citer J.G. Stell en 1999 (14) qui a formalisé la notion de graphe granulaire. Mais ses recherches n'ont pas été exploitées au cours de ce TFE.

III.1.2 Recherches antérieures et définitions pour la chaîne actuelle

La théorie des graphes a été détaillée dans le mémoire de TFE de Jean-Marc Beveraggi en 2018 (3). Néanmoins, certains points de définition ont été clarifiés et revus pour une meilleure correspondance avec ce TFE.

[6] Un article de Armutz publié en 1961 (12) fait office de référence concernant la théorie des graphes

❖ Définition d'un graphe :

Chaque graphe est constitué de deux types d'éléments, qui sont les arêtes (ou arcs) et les sommets (qui peuvent être des nœuds). Chaque arête est définie entre deux sommets. Chaque élément du graphe peut posséder une ou plusieurs composantes permettant de définir clairement chacun de ceux-ci.

❖ Graphe principal ou planaire :

C'est le graphe le plus simple à visualiser puisque les arêtes et sommets correspondent à des éléments qui existent naturellement. Dans notre cas, nous nous en servirons à l'échelle des parcelles, les arêtes étant les limites des parcelles, et les sommets étant les sommets des limites. Il pourra permettre de simplifier les éléments détectés (par exemple, en réduisant le nombre de segments qui aurait dû correspondre à une limite droite – voir Figure 17).

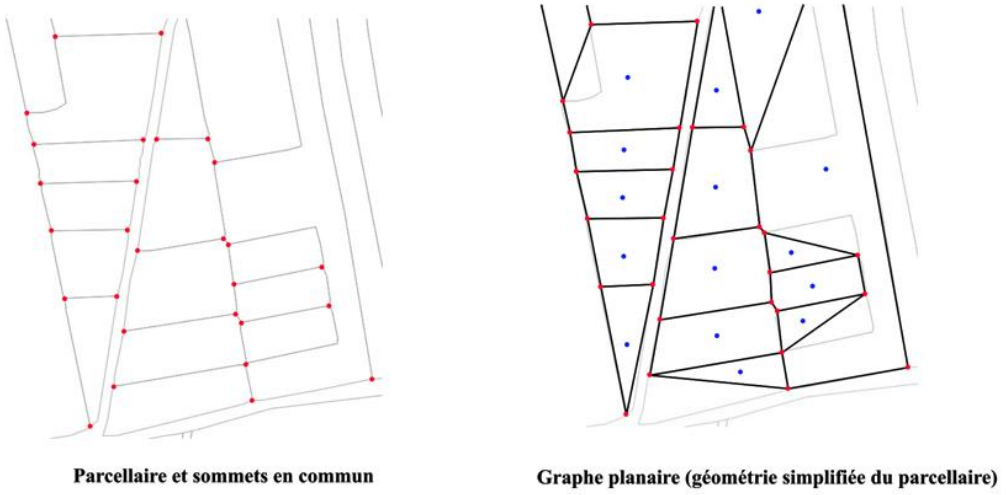


Figure 17 : Illustration du graphe planaire (Beveraggi, 2018)

❖ Graphe dual ou d'adjacence :

Ce second graphe est le graphe complémentaire au graphe planaire. Il définit les relations d'adjacence entre deux éléments de même niveau. Ici, le graphe d'adjacence sera le graphe reliant un point à l'intérieur d'une parcelle (le plus souvent le centroïde) à un autre ; ce seront les sommets du graphe ; et ils seront rejoints grâce aux relations de voisinage ; ce seront les arêtes (voir Figure 18).

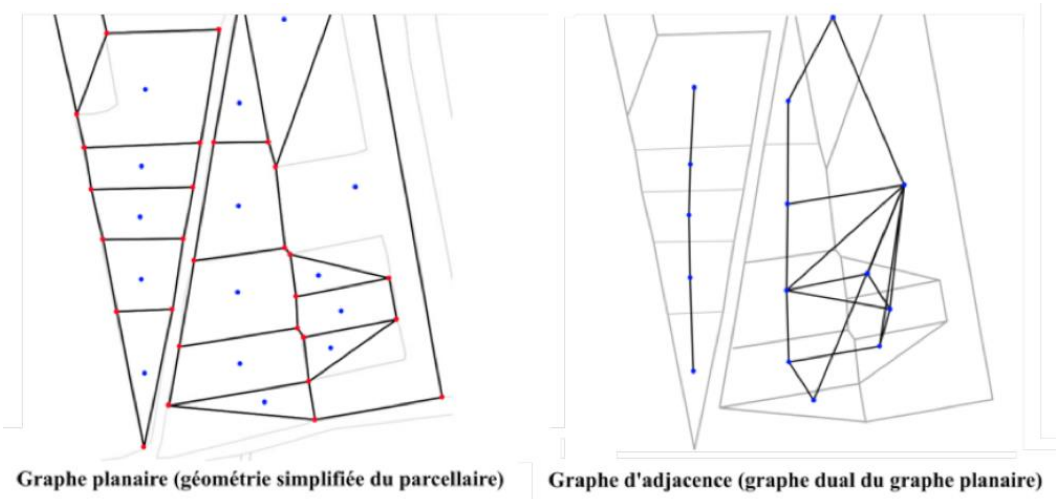


Figure 18 : Illustration du graphe d'adjacence (Beveraggi, 2018)

❖ Graphe temporel :

Ce graphe permet d'étudier les relations qui existent entre des parcelles situées au même endroit, mais à des époques différentes. Les sommets représentent les parcelles à une époque donnée et les arcs relient les différentes représentations des parcelles au cours du temps. Ce graphe mettra donc en évidence les changements d'état des parcelles ; la division ou la fusion (voir Figure 19).

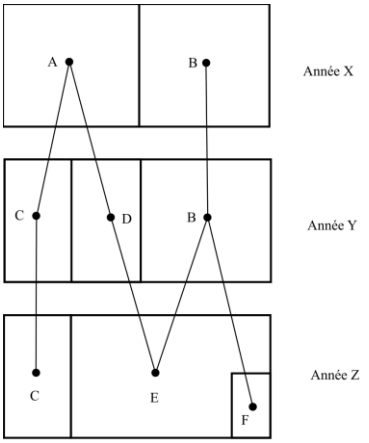


Figure 19 : Illustration du graphe temporel (Beveraggi, 2018)

❖ Graphe granulaire :

La notion de graphe granulaire permet de décliner des graphes à différentes granularités, notamment le niveau de détail. Dans notre cas, nous définirons la granularité qui existe entre les parcelles d'une même planche, puis celle qui relie les planches, formant une commune. Les sommets seront les centroïdes de chacun des ensembles de parcelles, et les arêtes, les relations de voisinage qui existent entre eux. Notons qu'un graphe granulaire peut être construit partiellement à partir d'un tableau d'assemblage (tableau représentant la répartition des planches dans une commune – voir la partie IV.1.1 pour un exemple).

III.2 L'utilisation de la théorie dans la chaîne de traitement

III.2.1 L'utilité des graphes dans le processus

Voyons les besoins nécessaires et l'utilité de chacun des graphes dans la chaîne de traitement, pour déterminer l'ordre des opérations à réaliser. Dans la suite, lorsque nous parlerons de position relative, nous parlons de la connaissance précise de la position spatiale d'entités par rapport à d'autres entités de même niveau (sans qu'il n'y ait nécessairement de géoréférencement).

- Le graphe planaire :

Comme dit dans la sous-partie précédente, ce graphe servira à simplifier les géométries. Nous ne détaillerons pas le mécanisme de simplification, qui n'aura pas de conséquence sur l'ordre des étapes dans la chaîne.

- Le graphe granulaire :

Nous définissons trois niveaux de granularité : le niveau 1 correspond à la commune, le niveau 2 la planche, et le niveau 3 correspond aux parcelles. Dans un second temps, on pourra ajouter des niveaux supérieurs, comme le département ou la région. Les relations, qui seront créées entre les éléments du niveau 2 seront réalisés à partir du tableau d'assemblage (qui existe pour chaque objet de niveau 1, les communes). Seul un opérateur pourra créer ce graphe, car dans une bonne partie des tableaux d'assemblage, il est difficile de détecter les limites des planches sans connaître leur forme à partir des planches prises individuellement.

Ce graphe sera un graphe d'adjacence, où chaque sommet du graphe correspondra à un ensemble de parcelles (le centroïde de chaque planche en l'occurrence). Cela permettra de mosaïquer chaque planche directement (sans passer par un géoréférencement préalable de chaque planche). Pour la définition des attributs des relations, il existe plusieurs possibilités, mais seuls les tests pourront déterminer leur valeur correcte.

- Le graphe temporel :

Ce graphe fait partie des données clés pour le laboratoire. Toutes les informations, fournies par celui-ci répondent parfaitement aux questions sur l'évolution du territoire. Ce graphe sera calculé à la fin du traitement des planches. Néanmoins, il est intéressant de s'en servir pour corriger d'éventuelles erreurs sur les polygones,

les parcelles évoluent fréquemment suivant une certaine logique, notamment concernant les formes des limites. En revanche, l'espace non cadastré varie très peu (sauf après la construction d'un ouvrage important, ou par correction d'une erreur passée sur la planche) ; en conséquence l'enveloppe globale (définie par l'ensemble des parcelles d'une planche) ne fluctue qu'exceptionnellement. De même, la fusion ou la division de parcelles modifient peu cette enveloppe.

La fabrication d'un graphe temporel pour chaque planche est donc intéressante pour réaliser des corrections, en particulier quand il s'agit des espaces non cadastrés. Il existe une relation directe avec le graphe d'adjacence, ce qui permettrait de corriger ce dernier automatiquement. Cependant, ce graphe inclut des données difficiles à obtenir au départ, à savoir les graphes et parcelles géoréférencées des autres années et les parcelles géoréférencées détectées sur la planche actuelle. Son utilisation s'insère dans la chaîne : entre le géoréférencement et le graphe d'adjacence, s'il est créé.

Ce graphe peut se constituer manuellement, dans le cas où l'opérateur doit aussi relever des points à l'intérieur des parcelles. Ainsi, il en profiterait pour définir « l'histoire des parcelles ».

- Le graphe d'adjacence :

Jean-Marc Beveraggi avait principalement travaillé sur ce graphe, puisqu'il a réalisé un programme Python pour le construire, avec les modules PySal et geopandas (15,16). À ce jour, il n'est pas question de modifier le fonctionnement de son programme, puisque le manque de temps ne nous permet pas de réaliser les tests nécessaires. Nous allons plutôt aborder la question de sa position et de son utilité dans la chaîne.

Tout d'abord, il est impossible de créer ce graphe avant que les parcelles n'aient été détectées, ce qui réduit fortement les possibilités de tirer profit de ce graphe. Toutefois, il peut être pertinent dans la phase de correction du mosaïquage.

Dans la chaîne, le graphe d'adjacence sera « étoffé » car des relations supplémentaires existent entre les parcelles au bord des planches et l'extérieur des planches, en raison de la présence d'éventuels traits sur la planche définissant un espace non cadastré qui séparerait les parcelles situées au bord de celles des planches voisines. Les extrémités de ces nouvelles relations sont d'une part les centroïdes des

parcelles au bord de la planche, et d'autre part associées aux segments qui appartiennent à l'enveloppe de la planche.

Les attributs justifiés par Jean-Marc Beveraggi lors de son TFE ne seront pas modifiés. Pour les nouvelles relations entre planches voisines, nous regardons s'il existe au moins cinq pixels (valeur arbitraire) appartenant à un ou plusieurs segments validés inclus dans le rectangle formé par les deux extrémités du segment étudié, et deux autres sommets dans la direction opposée à l'intérieur de la planche. Si c'est le cas, on ajoutera une connexion indirecte, sinon une connexion directe.

- Le graphe des contraintes :

C'est un graphe lié aux différentes règles régissant les planches cadastrales, avec des possibilités d'incertitudes, tel que le positionnement des numéros de parcelles devant se trouver à l'intérieur d'une parcelle. Il sera construit manuellement à l'analyse des documents, et pourrait servir à différentes étapes de la chaîne. Mais celui-ci n'aura pas d'influence sur l'ordre des étapes à réaliser (puisqu'il n'a pas besoin de données initiales).

III.2.2 Les scénarios possibles de la chaîne de traitement avec cet outil

Les différentes façons de construire chacun des graphes permet d'envisager plusieurs arrangements de la chaîne de traitement. Ces agencements, appelés « scénarios » dans la suite, sont divisés en trois blocs distincts (termes repris ensuite dans l'Annexe 6) :

- Les données initiales, correspondant aux éléments disponibles au début du traitement d'une planche (éventuellement grâce à une précédente exécution de la chaîne ou une intervention manuelle), et utilisables directement ;
- Les étapes immuables qui utilisent une ou plusieurs données nécessaires et suffisantes. Elles sont soit irréalisables car il manque une ou plusieurs données nécessaires, soit l'ajout d'autres données n'influence pas leur résultat ;
- Les étapes perfectibles auront des résultats dépendant des sorties du processus pour l'année en cours d'étude.

Parmi les données initiales pouvant être utiles à la chaîne, nous avons recensé les planches (obligatoires), les points de liaison (obligatoires), les résultats de la chaîne appliquée sur une autre planche (appelés « données autres planches », optionnelles), la position du centre (ou un point autre de l'intérieur, saisi par l'opérateur) de chaque parcelle

(optionnel) et le tableau d'assemblage (à étudier manuellement, optionnel). Il y a deux étapes immuables (à réaliser dans l'ordre), qui sont la détection des segments puis la détection des caractères (telle que nous l'avons décrite dans la section II.1.1). Enfin, les étapes perfectibles correspondent à la détermination des graphes (d'adjacence et éventuellement temporel), au géoréférencement et au mosaïquage. Notons que pour ces deux dernières étapes, la possibilité de connaître la relation entre les planches adjacentes pourrait venir modifier la chaîne de traitement définie précédemment, en inversant le géoréférencement et le mosaïquage.

En prenant en compte toutes ces réflexions, nous pouvons en déduire différents morceaux de scénarios :

- Nous considérons que la donnée initiale des centres des parcelles comprend le nom des parcelles. Si cette couche vectorielle est disponible, la détection des caractères ne sera pas effectuée. De plus, si nous disposons des résultats d'autres années, nous considérons que l'opérateur peut créer partiellement et simultanément le graphe temporel, selon la complexité des cas ;
- Le graphe temporel peut être réalisé, soit manuellement au début (voir ci-dessus), soit entre le géoréférencement et le graphe d'adjacence (sinon, il est inutile pour l'amélioration de la vectorisation ou irréalisable sans le géoréférencement), soit à la fin, car il sera essentiel à l'analyse des dynamiques territoriales ;
- Le graphe d'adjacence sera créé pour le mosaïquage uniquement, mais cela nécessite un minimum de données (parcelles détectées, nécessairement géoréférencées ou accompagnées du graphe granulaire) ;
- Le mosaïquage est réalisable avant le géoréférencement uniquement si nous connaissons les relations qui existent entre les planches, c'est-à-dire grâce au graphe granulaire. Cet échange permettrait de réduire le nombre de points de liaison à créer (étape manuelle) et d'éviter certaines erreurs topologiques (puisque la correction s'effectue juste après le mosaïquage. Au géoréférencement, les données seraient déjà très propres).

Au total, nous avons identifié 24 scénarios différents, dont 6 non réalisables. Parmi l'ensemble de ces scénarios, il existe six arrangements d'étapes perfectibles différents (réalisables ou non), qui sont détaillées en Annexe 6. Nous définirons un scénario comme

étant un ensemble {données + étapes immuables + étapes perfectibles} unique, et si plusieurs scénarios ont le même arrangement d'étapes perfectibles, alors le regroupement de ces scénarios forme un processus.

Parmi l'ensemble des processus, un seul sera appliqué à la chaîne de traitement finale. Néanmoins, dans un souci d'obtenir des résultats avant la fin du TFE, un scénario « provisoire » a dû être sélectionné, celui nommé PGAM (1) (voir Annexe 6) qui s'appuie uniquement sur la planche et les points de liaison, sans réaliser de détection de caractères. Le graphe d'adjacence, réalisé par Jean-Marc Beveraggi en 2018, a donc été introduit avant le mosaïquage, pour améliorer cette dernière étape.

La chaîne de traitement finale devrait inclure la détection de caractères, qui serait une véritable plus-value de la chaîne, et constituerait une véritable avancée vers l'automatisation complète de la chaîne. De plus, la lecture du tableau d'assemblage doit être pris en compte, car c'est une donnée déjà disponible et facile à transcrire en graphe granulaire. Enfin, ajouter le graphe temporel améliorerait tout le processus. Finalement, le meilleur processus semblerait donc être PGTAM ou PAMGT (voir Annexe 6), selon la position du mosaïquage et du géoréférencement. Notons que ces deux processus ne fonctionnent qu'avec les résultats d'une année précédente, ce qui n'est pas le cas des premières données qui seront traitées. Dans ce cas, on utilisera le PGAM (1) ou le PAMG (1) (voir Annexe 6).

IV Applications

IV.1 Présentation des données

IV.1.1 Principales caractéristiques du cadastre

L'histoire et les spécificités du cadastre ont déjà été largement développées par les trois précédents TFE cités précédemment (1–3). Néanmoins, nous allons rappeler quelques points importants ; la Figure 20 ci-dessous correspond au tableau d'assemblage de la commune étudiée, Aubigné-Racan.

Figure 20 : Tableau d'assemblage d'Aubigné-Racan (1850)

Le cadastre peut avoir subi plusieurs modifications plus ou moins importantes. D'abord, entre 1850 et 1972, de lourdes mutations (rénovation, remembrement, remaniement), ont provoqué sur certaines zones, des modifications profondes de sections et parcelles. La construction de grands axes (dès 1850 les voies ferrées) a le plus souvent créé de nouveaux espaces non cadastrés de grande superficie, modifiant parfois les frontières de certaines sections. Enfin, les modifications mineures (division ou fusion de parcelles) ont, à plus petite échelle, eu un impact sur le parcellaire. L'étude diachronique des planches est complexe, mais des similitudes existent entre les données multi-datées, et nous allons nous en servir pour appliquer la chaîne de traitement.

Au fil des TFE, différentes données ont été étudiées. Malheureusement, il a manqué une certaine coordination dans l'utilisation des données (zones géographiques différentes, années différentes…). Pour étudier l'avancée de la chaîne de traitement, nous allons utiliser

des planches d'années différentes sur la même zone géographique. Les planches appartiennent à la commune d'Aubigné-Racan. Ce sont les planches 009 (1813 – voir Figure 21), 114 et 115 (1850 – voir Figure 22), et 151 et 152 (1972 – voir Figure 23).

Figure 21 : Planche de 1813 étudiée (Planche 009, commune d'Aubigné-Racan)

IV.1.2 Planches de 1813 et 1850

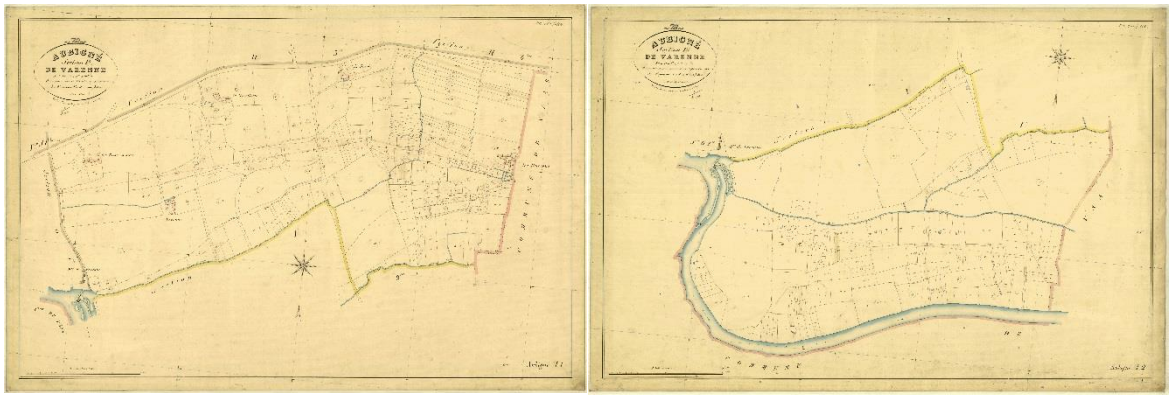


Figure 22 : Planches de 1850 étudiées (Planche 114 à gauche et 115 à droite, commune d'Aubigné-Racan)

Les planches de 1813 et 1850 sont souvent de mauvaise qualité. Jaunies, griffonnées, avec des marques de pliures, il est difficile (même manuellement parfois), d'en détecter les limites. De plus, selon la zone géographique, les planches peuvent avoir des caractéristiques variables : orientation et/ou échelle différente, présence d'agrandissements, voire même couleur du papier qui peut radicalement changer (de très clair à très foncé). Notons qu'en 1813 que certaines parcelles sont fusionnées, mais les limites n'ont pas été effacées (juste barrées, avec un symbole assez reconnaissable). Et pour une même zone géographique, la

transition entre 1813 et 1850 peut se traduire par la division d'une planche de 1813 en plusieurs planches de 1850.

Ces caractéristiques rendent le travail de vectorisation difficile. À ce jour, nous appliquons un prétraitement, pour éliminer l'effet jauni (aucun test n'ayant été réalisé pour les planches plus foncées), et supprimer des marques pouvant dégrader les résultats (grâce au seuillage par Hystérésis local modifié). L'épaisseur des lignes reste néanmoins constante sur toutes les planches d'une même année.

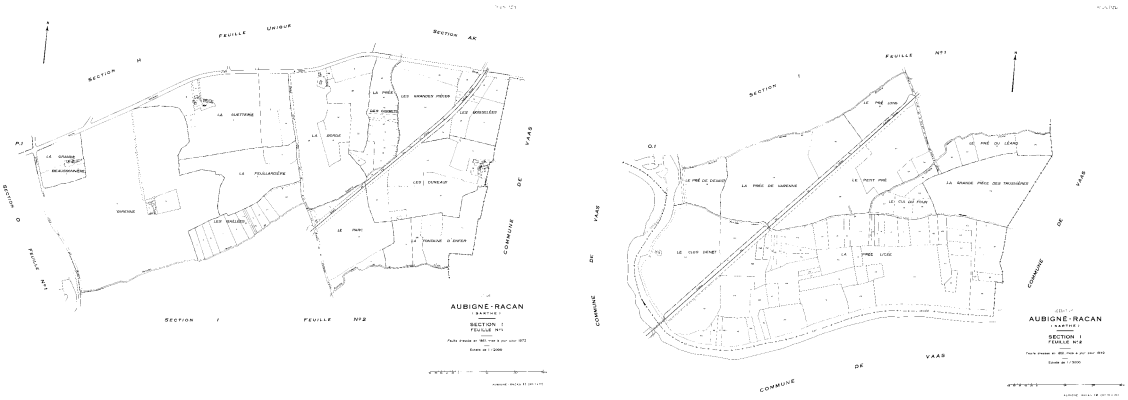


Figure 23 : Planches de 1972 étudiées (Planche 151 à gauche et 152 à droite, commune d'Aubigné-Racan)

IV.1.3 Planches de 1972

Par rapport à 1813 et 1850, les planches de 1972 sont de meilleure qualité et sont déjà codées en noir et blanc. Nous distinguons beaucoup plus facilement les caractères et les limites puisqu'il n'y a plus de marques de crayon ou d'écrits. Il existe des lignes d'épaisseur différente, qui servent à distinguer les lieux-dits (voir Figure 24). En dernier élément, il faut noter la résolution de la planche, puisque sa taille est plus grande que les planches de 1813 et 1850, ce qui peut rendre le traitement plus long selon le nombre de segments traités.

L'avantage de ces planches est la propreté des résultats obtenus. En effet, l'étude binaire de la planche suffit, ce qui peut permettre d'utiliser un algorithme de détection plus simple (voir la partie IV.2). Néanmoins, le post-traitement peut être conséquent.

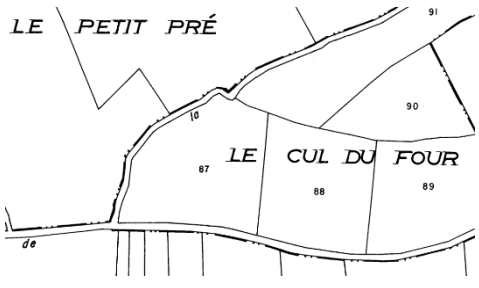


Figure 24 : Extrait de la planche 152 d'Aubigné-Racan, avec des lignes d'épaisseur différentes pour séparer les lieux-dits

IV.2 Les résultats de la vectorisation

Dans cette partie, nous nous intéresserons uniquement aux résultats issus de la première étape : la vectorisation. Ce préambule est nécessaire car les erreurs les plus importantes à la fin du traitement proviennent en majorité de la qualité des résultats de cette étape. De plus, ce sont, à ce jour, les seules données de type quantitatif qui sont exploitables pour comprendre les problèmes et savoir où les corriger.

Nous détaillerons deux types de comparaisons : les résultats en sortie de l'algorithme de détection, pour juger de la qualité du processus principal, et les sorties après l'application du post-traitement. Par rapport aux observations de Charlotte Odie en 2017 (2):

- Nous évaluerons uniquement le nouvel algorithme implémenté, LSD, et le HP. L'évaluation de la vectorisation par suivi de chemin par GRASS aurait pu apporter des informations complémentaires, mais la complexité de mise en œuvre et la volonté de s'affranchir au maximum de l'utilisation d'autres logiciels, limitent son intérêt.
- Nous omettrons les statistiques avancées par Charlotte Odie pour deux raisons essentielles : l'utilisation de planches cadastrales différentes et un calcul statistique faux.

Les algorithmes de détection permettent d'obtenir un ensemble de segments qui normalement, grâce au pré-traitement préalable, doit correspondre à des limites de parcelles. Pour évaluer statistiquement les résultats, nous définissons deux termes, déjà utilisés dans les précédents TFE :

- Le pourcentage de bonnes détections (PBD) est le pourcentage de pixels issus des segments vectorisés qui correspondent à une limite cadastrale, par rapport au nombre total de pixels appartenant à des limites cadastrales.
- Le pourcentage de fausses alarmes (PFA) est le pourcentage de pixels issus des segments vectorisés qui ne correspondent à aucune limite cadastrale par rapport au nombre total de pixel correspondant aux segments vectorisés.

À cela, nous ajouterons une estimation de la durée d'exécution réelle, du pré-traitement à la fin de la détection. Ce temps dépend du matériel utilisé, qui est le même pour chacune des exécutions réalisées. Ainsi, nous disposions d'un cluster de calcul composé de plusieurs serveurs (nœuds de calcul), parmi lesquels, deux ont été utilisés. Chacun des deux nœuds se compose de deux processeurs Intel® Xeon® de 2,20 GHz (totalisant 48 processus

(CPU)) et la RAM totale est de 62 Go. Précisons toutefois que seul un processus a été utilisé lors des calculs sur chacun des deux nœuds.

Chaque statistique a besoin d'une référence, c'est pourquoi, pour toutes les planches cadastrales utilisées, les parcelles ont été vectorisées manuellement, pour en ressortir les limites sous forme rasterisées ; cette image n'est utilisée que pour le calcul des statistiques. Cette réalisation a été chronométrée, et elle prend en compte les difficultés rencontrées lors de la détermination des parcelles, liées à la qualité des planches (pour les années 18XX).

Maintenant, avant de parler des résultats définitifs qui sont comparés avec les précédents algorithmes, nous allons détailler les paramètres utilisés dans le cadre de l'adaptation du LSD avec les données cadastrales.

IV.2.1 L'adaptation du LSD aux planches cadastrales

Avant de lancer la chaîne dans sa totalité, nous avons d'abord réalisé différents tests sur une partie ou l'ensemble des planches cadastrales utilisées par Charlotte Odie, pour juger le comportement du LSD. Nous avons modifié quelques paramètres du LSD, qui devaient être fixés selon l'article (4). Nous ne détaillerons l'influence que d'un seul des six paramètres du LSD, la tolérance sur l'angle du gradient, et ses conséquences sur la réalisation en plusieurs morceaux de l'algorithme. Les autres adaptations réalisées sur LSD ont réduit de façon importante le nombre de segments détectés. Sans avoir besoin de chiffrer les résultats obtenus, nous avons remarqué à la fin du nouveau post-traitement l'absence quasi complète des limites à détecter sur les planches car soit aucun segment n'a été créé, soit les segments étaient complètement en dehors des limites.

Concernant la modification de la tolérance sur l'orientation du gradient, la valeur fixée était de 22.5°. D'autres valeurs ont été testées comme 90°, 45° ou 18°, mais aucune de ces valeurs n'apportait de résultats satisfaisants. Seule la valeur 30° pouvait être discutée. Pour la réalisation du LSD en plusieurs morceaux, plusieurs découpages ont été testés, en deux ou soixante-quatre morceaux pour les années 1800, mais, comme pour la tolérance, les résultats ou le temps d'exécution n'étaient pas admissibles. La seule limite fixée était de garder un nombre correspondant à une puissance de 2, pour que le calcul des partitions de l'image soit facilement réalisé par le programme. Ce sont donc les valeurs 1 et 4 (c'est-à-dire 1 ou 16 morceaux) pour les années 1800 et 8 (c'est-à-dire 64 morceaux) pour l'année 1972, avec des planches de plus grande taille (réalisation du LSD à 64 reprises) qui ont été retenues.

Paramètre de Découpage / Tolérance sur l'orientation du gradient	**1** Correspondant à 1 exécution du LSD sur l'ensemble de l'image étudiée	**4** Correspondant à 16 exécutions du LSD, sur chacun des 16 morceaux de taille égale
30°	NON RÉALISÉ Sur des morceaux d'images, la détection donne une combinaison des inconvénients des autres tests.	Temps d'exécution : 1 h Bonnes détections : 88 % Fausses alarmes : 38 %
22.5°	Temps d'exécution : 8 h Bonnes détections : 88 % Fausses alarmes : 34 %	Temps d'exécution : 1 h Bonnes détections : 88 % Fausses alarmes : 35 %

Tableau 1 : Statistiques après une dizaine d'exécution sur la planche 008 (1813) d'Aubigné-Racan

Paramètre de Découpage / Tolérance sur l'orientation du gradient	**1** Correspondant à 1 exécution du LSD sur l'ensemble de l'image étudiée	**8** Correspondant à 64 exécutions du LSD, sur chacun des 64 morceaux de taille égale
30°	NON RÉALISÉ Sur des morceaux d'images, la détection donne une combinaison des inconvénients des autres tests.	Temps d'exécution : 30 min Bonnes détections : 87 % Fausses alarmes : 9 %
22°5	Temps d'exécution : 30 h Bonnes détections : 89 % Fausses alarmes : 9 %	Temps d'exécution : 30 min Bonnes détections : 89 % Fausses alarmes : 9 %

Tableau 2 : Statistiques après une dizaine d'exécution sur la planche 155 (1972) d'Aubigné-Racan

Les temps d'exécution calculés montrent qu'il vaut mieux lancer LSD sur plusieurs morceaux d'une image que sur l'image entière pour une qualité de résultat équivalente. Dans la suite, nous privilégierons donc une exécution multiple du LSD.

En ce qui concerne la tolérance sur l'orientation du gradient, il est difficile d'en tirer des conclusions. Néanmoins, il est nécessaire de choisir un paramètre unique pour éviter de complexifier la chaîne, et pour la garder la plus compréhensible. Ainsi, nous avons décidé de prendre la valeur 22.5° pour une raison principale : la petite différence de pourcentage de trois points en fausses alarmes pour l'année 1813.

Les paramètres pour LSD fixés pour la suite sont donc :

- Tolérance de l'orientation du gradient fixée à 22.5° ;
- Découpage de l'image pour l'exécution du LSD fixé à 4 pour 18XX (correspondant à un découpage 4x4) et 8 pour 19XX (découpage 8x8).

IV.2.2 Application de l'algorithme LSD définitive avec le pré-traitement

Nous allons maintenant traiter les résultats issus de la détection, en utilisant soit LSD soit HP. Le Tableau 4 recense les statistiques sur l'exécution de la chaîne, dont les paramètres en entrée sont :

- Pour le pré-traitement, lorsque que c'est LSD qui est utilisé pour la détection, nous effectuerons un seuillage par hystérésis avec comme facteur pour le calcul du seuil bas (1.10 fois la moyenne locale). La valeur prise pour les pixels au-dessus du seuil bas correspondra au maximum local (voisinage 3x3). Avec HP, nous effectuerons une squelettisation de l'image. Pour l'ensemble des planches, nous appliquons un masque correspondant au contour grossier du groupe de parcelles.

- Pour les paramètres de détection, lorsque que c'est la transformée de Hough Probabiliste qui est choisie, nous utiliserons les mêmes paramètres qui ont été déduits par Charlotte Odie en 2017 (2) ; voir le Tableau 1. Concernant LSD, voir la partie IV.2.1.

	threshold	line_length	line_gap	pas_theta
1813	15	30	15	0.5°
1850	5	20	15	0.5°
1972	5	20	20	0.5°

Tableau 3 : Paramètres utilisés pour la Transformée de Hough Probabiliste (définition de paramètres dans l'Annexe 4)

- Enfin, nous ajoutons un paramètre de précision finale valant un pixel. Ce paramètre permet d'obtenir deux statistiques différentes. D'une part, nous comparons directement la référence manuelle et la détection ; ce seront les statistiques « précises ». D'autre part, nous comparons les mêmes données, en considérant les pixels voisins d'une limite comme appartenant à une limite ; ce seront les statistiques « tolérées ». Cette précision permet de prendre en compte les limites de calcul de Python, mais aussi les segments qui, à cause de la mauvaise qualité des planches, seraient légèrement décalés de la réalité terrain et seraient considérés comme faux.

		1813 Planche 009	1850 Planches 114 | 115	1972 Planches 151 | 152
Temps d'exécution	HP	0 h 08	0 h 09 | 0 h 08	0 h 03 | 0 h 03
	LSD	2 h 21	1 h 12 | 0 h 58	1 h 56 | 1 h 48
Nombre Segments	HP	9 020	12 783 | 9 339	7 678 | 5 050
	LSD	12 948	10 453 | 7 547	10 770 | 8 779
Bonnes détections	HP précis	83,05 %	94,68 % | 98,60 %	96,55 % | 96,99 %
	HP tolérée	84,59 %	95,68 % | 98,92 %	99,39 % | 99,44 %
	LSD précis	80,42 %	93,78 % | 96,69 %	99,01 % | 98,72 %
	LSD tolérée	87,21 %	97,53 % | 98,91 %	99,88 % | 99,97 %
Fausses alarmes	HP précis	27,54 %	45,19 % | 40,77 %	41,38 % | 25,62 %
	HP tolérée	18,17 %	38,20 % | 33,66 %	37,59 % | 23,18 %
	LSD précis	37,53 %	48,65 % | 49,90 %	40,99 % | 31,86 %
	LSD tolérée	31,87 %	43,92 % | 47,57 %	37,68 % | 29,29 %

Tableau 4 : Résultats à l'issue de la vectorisation, c'est-à-dire sur les segments détectés

Sur plusieurs points, les algorithmes s'opposent. Par exemple, concernant le temps d'exécution, il est évident que HP est beaucoup plus rapide que LSD. En moyenne, HP aura besoin de quelques minutes, alors que LSD a besoin d'au moins 1 heure. Concernant les pourcentages obtenus, tout dépend de la planche étudiée. Ainsi, pour les années 18XX, HP sera plus précis (selon les PBD précises et les PFA) mais LSD sera meilleur (selon les PBD tolérées). Il faut quand même faire attention au nombre de segments car s'il est important, cela signifie souvent qu'il y a plusieurs segments détectés alors qu'il aurait dû y en avoir qu'un seul. Ce nombre est plus important pour le HP en 1850 et l'inverse pour 1813. Pour l'année 1972, LSD prend un net avantage malgré un nombre de segments plus élevé qui influe nécessairement ces statistiques.

Ces observations prouvent qu'il s'agit de deux algorithmes très différents. Néanmoins, il est difficile de savoir quel est le meilleur algorithme de détection. HP est certes plus rapide, mais il est non déterministe, c'est-à-dire qu'il va produire des résultats différents à chaque exécution, alors que les données utilisées sont les mêmes. Il faut aussi faire attention à la qualité des résultats, car HP fournit un faisceau de segments pour une même limite à détecter (voir Figure 25). Ceci biaise les statistiques, car il y a nécessairement plus de pixels qui rentrent en compte. De l'autre côté, LSD fournit des résultats très corrects selon les PBD et PFA, mais les deux mille segments détectées en plus en moyenne et le temps d'exécution peuvent faire réfléchir à son efficacité.

Ces observations ne permettent pas d'envisager un algorithme plutôt qu'un autre. De plus, les segments détectés ne sont pas le résultat final demandé, car l'objectif est d'obtenir des parcelles. C'est le post-traitement qui nous permettra de faire le meilleur choix.

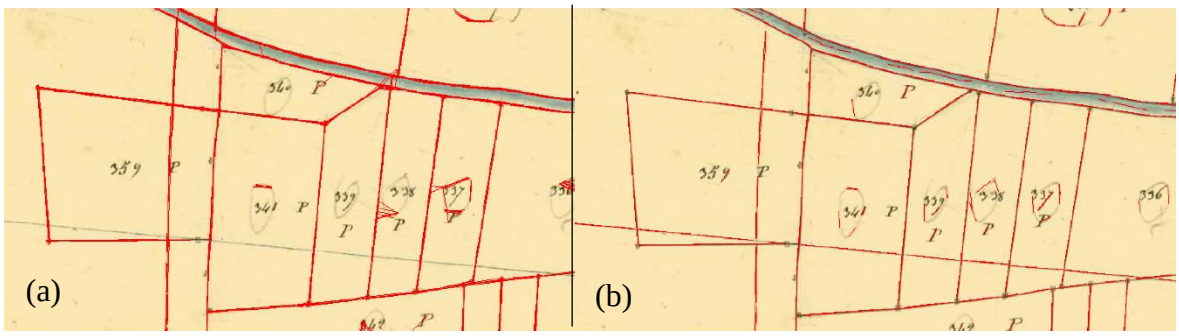
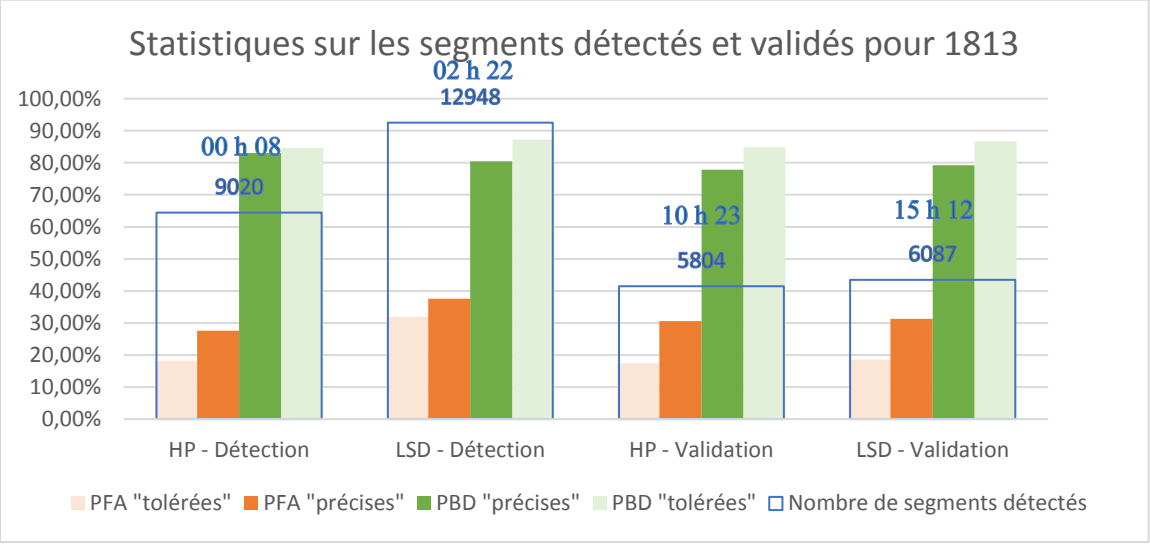
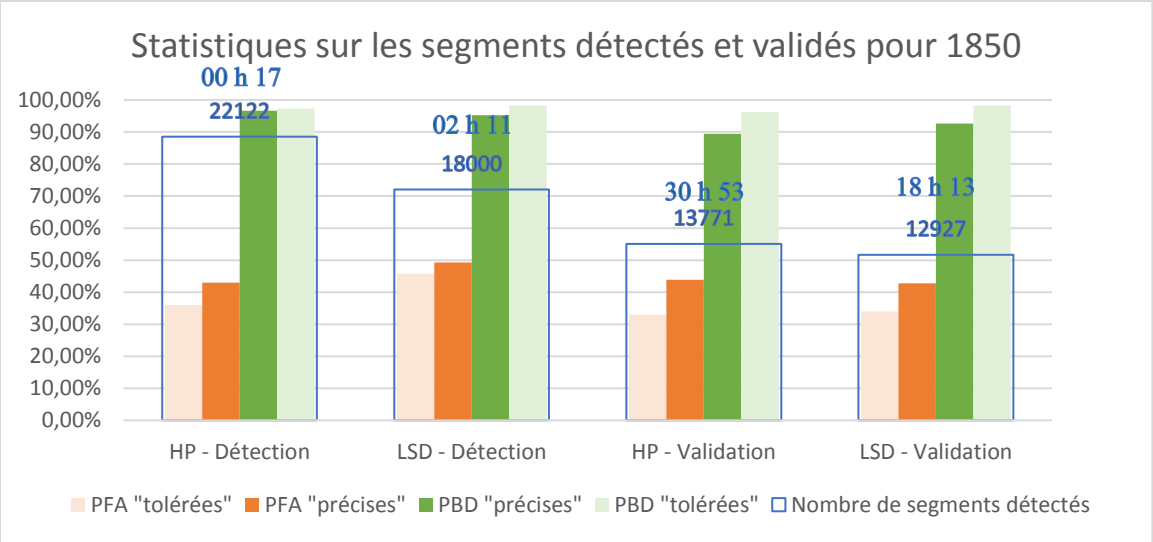


Figure 25 : Exemple de sortie à la détection sur la planche 115, avec en (a) HP et en (b) LSD

IV.2.3 Les résultats à l'issue du post-traitement

Comme expliqué dans la partie II.3.3, le post-traitement est séparé en deux parties. Nous allons donc d'abord voir les statistiques à l'issue du post-traitement n°1 (validation des segments), qui correspondent à des pourcentages sur les segments dits validés, puis nous étudierons les statistiques à l'issue du post-traitement n°2 (création des polygones), les plus importantes, qui apportent des détails sur les parcelles détectées.





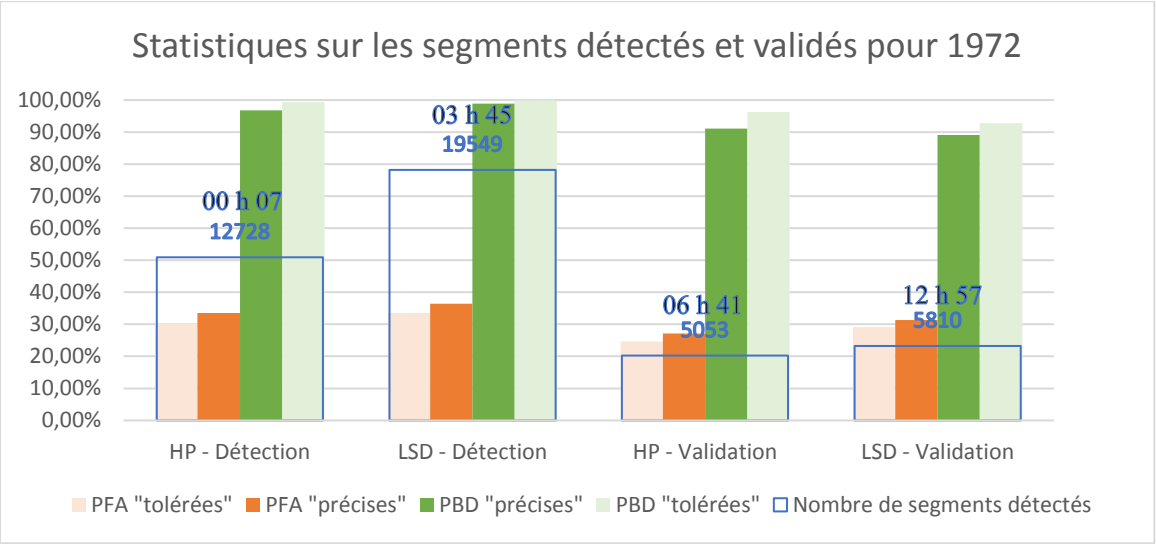


Figure 26 : Statistiques sur les segments détectés (à la fin de la détection) et validés (à la fin du post-traitement n°1)

D'abord de façon générale, on observe que le nombre de segments diminue de façon très importante après la phase de validation (parfois divisé par 4), et que les PBD (précises et tolérées) sont stables, alors que les PFA baissent de cinq à dix points en moyenne, quel que soit l'algorithme de détection utilisé. Cela montre l'efficacité du post-traitement, qui simplifie et garde presque tous les bons segments. En revanche, le temps d'exécution enfle. Cela s'explique notamment par la fonction *prolongements*, qui valide quasiment l'ensemble des couples de segments pouvant être formés. D'autre part, la validation demande de transformer en raster les segments, une opération longue qui ralentit considérablement le traitement.

Si nous nous intéressons à chaque algorithme individuellement, des remarques sont à souligner. Plus de segments ont été traités et supprimés avec LSD plutôt qu'avec HP. Les PBD et les PFA varient de la même façon. LSD est donc plus efficace avec les planches des années 18XX. Pour 1972, les pourcentages obtenus et le temps d'exécution sont largement plus favorables à HP plutôt qu'à LSD. Il semble donc plus intéressant d'appliquer HP pour les cadastres plus récents. Pour les années 18XX, en revanche, les observations montrent que LSD donne de meilleurs résultats par rapport aux PBD et au nombre de segments (200 segments d'écart pour 1813 sur les 6000 validés). L'écart du temps d'exécution est moins évident. Une des explications réside en la qualité des résultats qui peut provoquer l'apparition d'une boucle infinie lors de l'une des sous-étapes de la vectorisation. Ce problème a en partie été réglé par le biais de la suppression de cette sous-étape après 10 minutes de non-progression du pourcentage affiché.

Ainsi, nous commençons à voir quels algorithmes utilisés, mais le questionnement sur les parcelles finalement détectées est obligatoire. La Figure 27 présente les statistiques sur les parcelles qui ont été finalement détectées ou non. Dans cette figure, nous appellerons :

- ❖ Les parcelles « correctes », les parcelles de référence dont l'aire est représentée à 95 % par un unique polygone détecté ;
- ❖ Les parcelles « divisées », les parcelles de référence dont l'aire est représentée à 95 % par plusieurs polygones détectés ;
- ❖ Les parcelles « englobées », les parcelles de référence dont l'aire est incluse dans un polygone qui contient d'autres parcelles ;
- ❖ Les parcelles « partielles », les parcelles de référence dont l'aire n'est pas représentée à 95 % par un ou plusieurs polygones détectés ;
- ❖ Les parcelles « non détectées », les parcelles de référence dont l'aire n'a pas été détectée par au moins un polygone ;

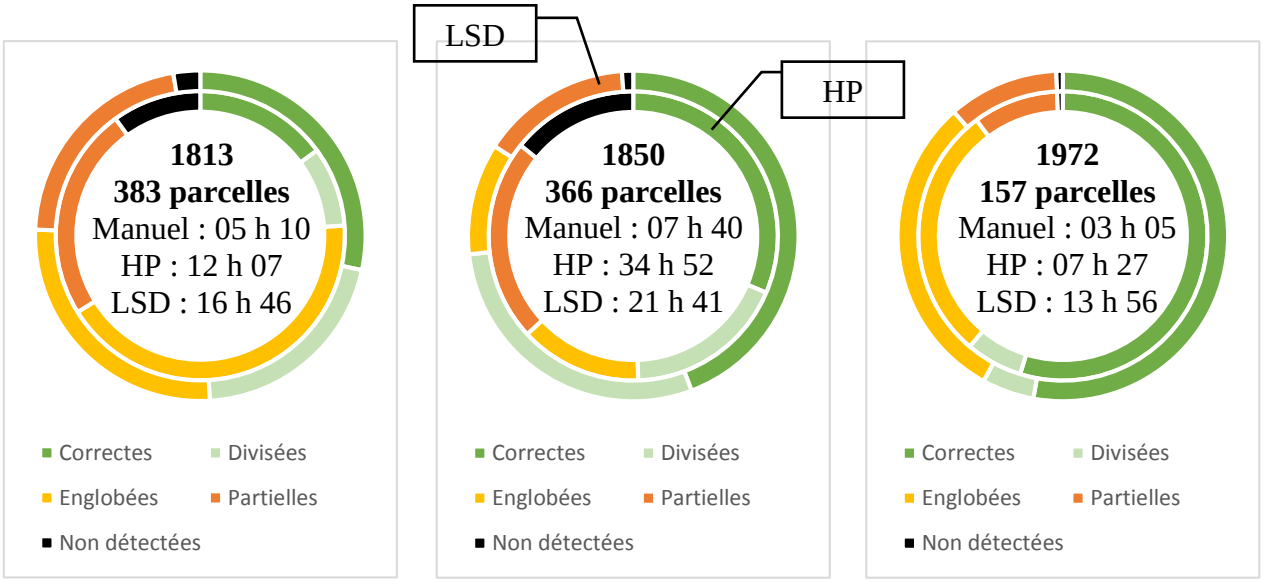


Figure 27 : Statistiques concernant les parcelles de référence (dessinées manuellement), avec, pour le cercle intérieur, les statistiques avec HP, et le cercle extérieur, les statistiques avec LSD

Avec ces cinq définitions, nous observons beaucoup mieux les différences entre les différents algorithmes. Par exemple, plus la planche est récente, plus la qualité est meilleure (sur l'ensemble des statistiques calculées). Ensuite, les résultats obtenus permettent clairement d'identifier l'algorithme le plus performant pour ces planches, dans l'optique de l'étendre sur toutes les années. Pour cela, en plus des parcelles correctes, nous allons aussi nous intéresser aux parcelles « divisées ». Le fait que ces parcelles soient morcelées signifie

que leur contour est déterminé. Ainsi, lors de la modification manuelle, une fusion suffira. La proportion des parties « vertes » (parcelles correctes et parcelles divisées) du graphique est donc un critère de qualité de l'algorithme.

Pour 1813, bien que HP soit plus rapide, les statistiques sont médiocres, surtout concernant la proportion de la partie « verte » qui correspond à 23,8 % des parcelles à détecter. Ainsi, malgré un processus qui dure 4h39 supplémentaires, et la statistique de la partie « verte » qui atteint difficilement les 50 %, LSD sera sûrement l'algorithme de détection le plus performant.

Pour 1850, nous avons presque le même constat, mais LSD est beaucoup plus rapide. Le temps d'exécution varie énormément selon plusieurs paramètres tel que le nombre de segments étudiés, le nombre de parcelles détectées, leurs répartitions et leurs tailles. Par exemple, pour 1850, si le temps pour HP est de 34 heures pour la totalité de l'exécution, en réalité, les deux tiers de ce temps ont été consacrés à l'étude de la planche 114 (l'autre tiers pour la planche 115). Il est difficile de trouver l'origine précise de cette différence. Seule une étude approfondie du rapport et d'autres tests résoudront ce problème. Dans tous les cas, LSD sera à priori le meilleur algorithme, puisque la partie « verte » atteint 73,2 % (contre 49,5 % pour HP).

Pour 1972, les statistiques sont proches. Celle de la partie « verte » atteint 61,1 % pour HP (96 parcelles sur 157) contre 58,0 % pour LSD (91 parcelles sur 157). Ces pourcentages ne donnent aucun avantage clair. Le temps d'exécution est plus déterminant puisque LSD le fait quasiment doubler. C'est pourquoi, nous pensons que pour 1972, le HP est suffisant.

Ces dernières observations ont permis de mettre en avant l'utilité du LSD dans deux cas sur trois, qui sont les plus intéressants car les plus problématiques. La question se posera lors de la généralisation sur l'ensemble des planches. On remarque que LSD est plus efficace sur les planches les plus anciennes. Mais, les résultats étant calculés sur un échantillon microscopique (5 planches sur les plus de 160, rien que sur Aubigné-Racan), nous ne pouvons pas généraliser cette fiabilité. D'autres limites de la chaîne sont expliquées dans le paragraphe IV.3.2.

IV.3 Exécution de la chaîne de traitement complète

IV.3.1 Résultats de la chaîne de traitement totalement automatique

Comme évoqué dans la section I.2, le code a été repensé. L'exécution se réalise d'un seul tenant, avec la possibilité de supprimer une des étapes si nous n'en avons pas besoin.

Le géoréférencement a pu être implémenté sans problème, en l'appliquant en plus sur la couche des limites en bordures de planches. Les résultats obtenus sont de même qualité que ceux obtenus par Maïté Fahrasmane au cours de son TFE (1), car le calcul ne dépend que des points d'appui et de l'image (qui n'ont pas été modifiés depuis).

Pour le mosaïquage, nous avons repris que partiellement la méthodologie de Jean-Marc Beveraggi, puisque les résultats obtenus n'ont pas été créés de la même façon, et nous disposons de données supplémentaires créées par la nouvelle vectorisation. De plus, nous avons décidé de ne pas réaliser le graphe d'adjacence de la planche, inutile pour la correction du mosaïquage, pour ne nous intéresser qu'au graphe d'adjacence entre les planches, qui permettra d'ajouter ou supprimer des espaces non cadastraux entre les planches. Mais un gros problème est survenu lors de la fabrication du graphe d'adjacence avec le module PySal sur Python : pour le fabriquer, aucune superposition, même entre des polygones distincts, ne peut être accepté. Or, en raison du géoréférencement planche par planche, il est impossible de passer outre ce problème, comme nous pouvons le voir dans la Figure 28.

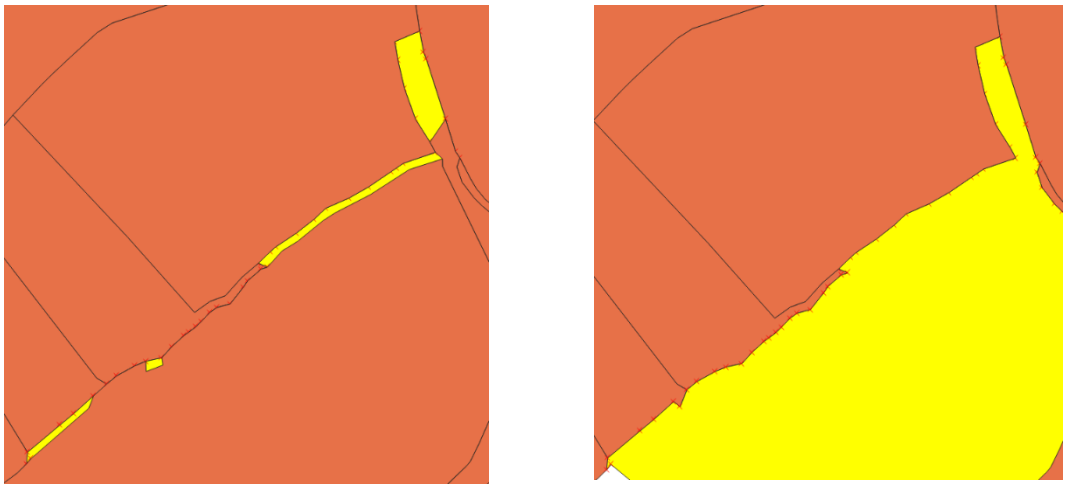


Figure 28 : Exemple de superposition à corriger avec en jaune, une seule et même parcelle, occultée par d'autres parcelles sur l'image de gauche

Ceci était déjà un problème rencontré par Jean-Marc Beveraggi, qui l'avait résolu grâce à ses corrections topologiques. Nous sommes tout de même obligés de garder les superpositions pour garantir la meilleure correction pour le mosaïquage. Mais, d'autres erreurs sont apparues, en particulier de type « Self-intersection ». Ce type d'erreur se réalise

normalement lorsqu'un élément géométrique (ou un groupe de géométries) s'intersecte avec lui-même. Ceci ne devrait pas arriver car les fonctions utilisées précédemment dans la vectorisation et le géoréférencement ont été créées dans le but d'éviter les erreurs topologiques. Ce qui est plus surprenant, c'est que ce type d'erreur apparaît pour presque l'ensemble des données géoréférencées, car elles s'affichent alors que ce ne sont que des sommets communs entre les polygones. Cette erreur, bien que détectée par la validation de Shapely, n'est pas corrigeable par ce même module, qui ne dispose pas de fonctions de correction globale (nous utilisons par exemple un buffer de 0 mètre), et pire encore, la tolérance utilisée dans les fonctions topologiques serait moins restrictive que la tolérance dans les fonctions de validation (les fonctions de validation sont donc beaucoup plus précises que les fonctions topologiques). En conséquence, après avoir testé plusieurs méthodes de correction (utilisation de la fonction *split* ; suppression des topologies incorrectes…), aucune ne permettent d'obtenir des données nécessaires au graphe d'adjacence.

Quelques corrections ont permis d'améliorer les résultats, comme la réunion des points similaires, une diminution de la précision des coordonnées (réduite au dixième de millimètre près, en supposant que les limites de Shapely se situent à 10^{-4} mètre près), ou l'utilisation de la fonction *snap* de Shapely (11), permettant d'adapter les polygones à leur environnement (en ajoutant des points là où il devait y en avoir – voir Figure 29).

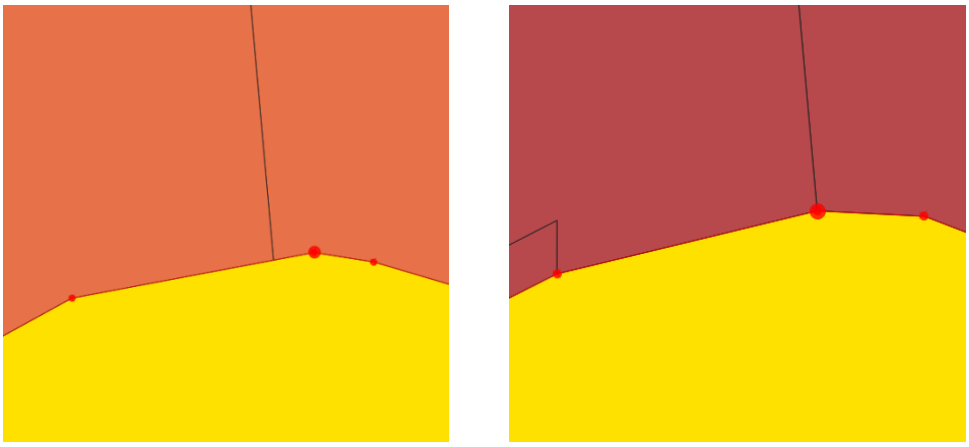


Figure 29 : Illustration de la fonction *snap* de Shapely, permettant d'ajouter un point sur le contour du polygone jaune, au niveau de l'extrémité du segment vertical (les autres évènements comme la suppression d'un des points et la création de segments sont dûs à des étapes précédentes)

IV.3.2 Observations et limites des résultats

En rassemblant toutes les observations depuis le début de la partie IV, le résultat est très encourageant pour la suite. Malgré des problèmes liés au mosaïquage, le nouvel algorithme de vectorisation implémenté a su apporter ses avantages pour les planches de 1813 et 1850. De plus, le post-traitement que nous avons repensé, a amélioré encore les

résultats. Malgré un bon géoréférencement, c'est le mosaïquage qui a plus de difficultés à s'exécuter. Peut-être que l'une des modifications sera de vérifier ces topologies à la fin de la vectorisation, voire de changer de module de traitement des topologies pour le mosaïquage. Mais il faut aussi mettre en œuvre une augmentation significative du nombre de tests.

En plus des réalisations citées précédemment, d'autres ont été effectuées sans en faire mention dans les paragraphes précédents, à cause de la limite des tests réalisés. Par exemple, il est fort intéressant d'étudier le paramètre min_pourcent, qui permet de gérer le pourcentage de pixels valides à obtenir pour garder un segment. Ce pourcentage peut avoir une forte influence sur les résultats (de plus ou moins 10 points en PBD ou PFA). De plus, ce paramètre a été modifié par rapport aux valeurs initiales déterminées par Charlotte Odie (2). Ce sont la réalisation de plusieurs tests avec LSD et les planches traitées par Charlotte Odie qui ont permis de passer à un pourcentage qui augmente si le segment est plus long (pour Charlotte Odie à l'année 1850), à un pourcentage qui, au contraire, ici, diminue si le segment est plus long (dans notre cas). Ainsi, min_pourcent vaut aujourd'hui ((100,0.80),0.60)[7] pour les années 1813 et 1850 et ((200,0.90),0.60) pour 1972.

Pour terminer, l'application à d'autres planches (correspondant à des territoires caractérisés par des occupations de sol, des reliefs ou produites par des méthodes différentes) est nécessaire car la chaîne dépend énormément de la donnée initiale. Chaque test réalisé sur une planche différente peut faire varier de beaucoup le PBD et le PFA, d'une planche à une autre, de la même année ou non. La chaîne de traitement va donc se confronter à beaucoup de problèmes à cause de la diversité des dessins. Pour autant, l'ajout des autres graphes (granulaire, temporel et des contraintes), la détection de caractères avec éventuellement le deep-learning, permettront de l'utiliser pour d'autres types ou configurations de communes. Aujourd'hui, seule l'étape manuelle, toujours indispensable, a une part encore très importante dans le bon déroulement du géoréférencement et surtout du mosaïquage.

Les données finales obtenues sont difficilement utilisables, comme on peut voir en Annexe 8. Mais pour une première exécution complète de la chaîne sans intervention humaine, les résultats sont très encourageants.

[7] Cela signifie que pour les segments inférieurs à 100 pixels, le pourcentage de validation des pixels est de 80%, et si la longueur du segment est supérieure, alors le pourcentage est de 60%. On favorise les segments plus longs, qui ont plus de chance de correspondre à des limites que les plus courts. Voir l'Annexe 4.

Conclusion et perspectives

Inscrite dans la continuité des travaux précédents, la chaîne de traitement de vectorisation des planches cadastrales a été améliorée à différents niveaux.

D'abord, la mise en forme algorithmique laisse place à des tests plus efficaces et rapides. Ajouté à cela, l'enregistrement du déroulement de la chaîne permet d'obtenir des statistiques précises et difficilement biaisables. Ensuite, le codage du nouvel algorithme de détection a apporté une grande évolution en termes de temps d'exécution et de qualité des résultats en particulier pour les planches anciennes. Enfin, les réflexions sur la théorie des graphes ont abouti à la définition du scénario de la chaîne de traitement finale.

Aujourd'hui et pour la première fois en cinq ans, nous obtenons des résultats après l'exécution complète de la chaîne, résultats qui sont par ailleurs meilleurs que les précédents travaux. Avec le perfectionnement du programme, il est maintenant beaucoup plus simple d'améliorer les sorties en se concentrant sur les étapes posant des problèmes ou en modifiant la valeur de quelques paramètres importants. Les tests seront donc plus simples à mettre en œuvre.

Beaucoup d'améliorations peuvent encore être apportées. Outre la modification éventuelle de quelques étapes (en particulier sur le post-traitement de la vectorisation et le mosaïquage), le programme peut s'enrichir de nouvelles phases. La détection de caractères tels que les numéros de parcelles permettrait de connaître le nombre de parcelles à extraire et leurs situations. Il faudrait utiliser concrètement les graphes dans la chaîne, en particulier le graphe granulaire qui corrigera grandement les vecteurs calculés. Le calcul du graphe temporel, ajouté à la création de la base de données, permettra réellement de commencer l'étude des territoires. Enfin l'ajout d'une interface graphique permettrait de rendre plus abordable et facilement utilisable la réalisation de la vectorisation, au sein du laboratoire ou même à l'extérieur dans le cadre d'une diffusion de l'outil.

Bibliographie

1. Fahrasmane M. Géoréférencement de cadastres anciens vectorisés au moyen de logiciels libres: application sur les communes de Vaas et Aubigné-Racan (Sarthe). Mémoire d'ingénieur, Le Mans, France : CNAM - ESGT, 2016, 76 p.

2. Odie C. Vectorisation semi-automatique de planches scannées du cadastre ancien. Mémoire d'ingénieur, Le Mans, France : CNAM - ESGT, 2017, 68 p.

3. Beveraggi J-M. Mosaïquage de données géoréférencées du cadastre ancien – vers une approche basée sur les graphes. Mémoire d'ingénieur, Le Mans, France : CNAM - ESGT, 2018, 55 p.

4. Grompone von Gioi R, Jakubowicz J, Morel J-M, Randall G. *LSD: a Line Segment Detector*. Image Process Line, [en ligne], 2012, 2. Disponible sur : http://www.ipol.im/pub/art/2012/gjmr-lsd/?utm_source=doi. (consulté le 20 févr 2019)

5. Cadastre. In : Direction interministérielle du numérique et du système d'information et de communication de l'état. Plateforme ouverte des données publiques françaises, [en ligne]. Disponible sur : https://www.data.gouv.fr/fr/datasets/cadastre/. (consulté le 24 mai 2019)

6. Index des planches cadastrales récentes des parcelles jamais vectorisées. In : Direction interministérielle du numérique et du système d'information et de communication de l'état. Données cadastrales ouvertes, [en ligne]. Disponible sur : https://cadastre.data.gouv.fr/data/dgfip-pci-image/latest/tiff/feuilles/. (consulté le 24 juin 2019)

7. Arteaga MG. Historical map polygon and feature extractor. In : Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction - MapInteract '13, Orlando, Florida, 2013. ACM Press, p. 66-71.

8. Cura R, Dumenieu B, Abadie N, Costes B, Perret J, Gribaudi M. *Historical collaborative geocoding*. ArXiv170307138 Cs, [en ligne], 2017. Disponible sur : http://arxiv.org/abs/1703.07138. (consulté le 24 mai 2019)

9. Index (All functions, classes, terms of skimage). In : scikit-image development team. scikit-image - image processing in python, [en ligne]. Disponible sur : https://scikit-image.org/docs/stable/genindex.html. (consulté le 4 mars 2019)

10. The Gamma function. In : Viktor T. Toth. Programmable Calculators, [en ligne]. Disponible sur : http://www.rskey.org/CMS/the-library/2-uncategorised/11-the-gamma-function. (consulté le 20 févr 2019)

11. The Shapely User Manual. In : Gillies S. Shapely 16 documentation, [en ligne] 2018. Disponible sur : https://shapely.readthedocs.io/en/stable/manual.html. (consulté le 23 mai 2019)

12. Amstutz P. *Théorie des graphes et ses applications de C. Berge*. Ann Télécommunications, [en ligne], 1961, 16, 1. Disponible sur : https://link.springer.com/article/10.1007/BF03020407. (consulté le 28 juin 2019)

13. Rodier X, Le Couédic M, Hautefeuille F, Leturcq S, Jouve B, Fieux E. From Space to Graphs to Understand Spatial Changes Using Medieval and Modern Fiscal Sources. In : Verhagen P, Earl G, (éd.), Archaeology in the Digital Era. Amsterdam : Amsterdam University Press, 2014, p. 420-7.

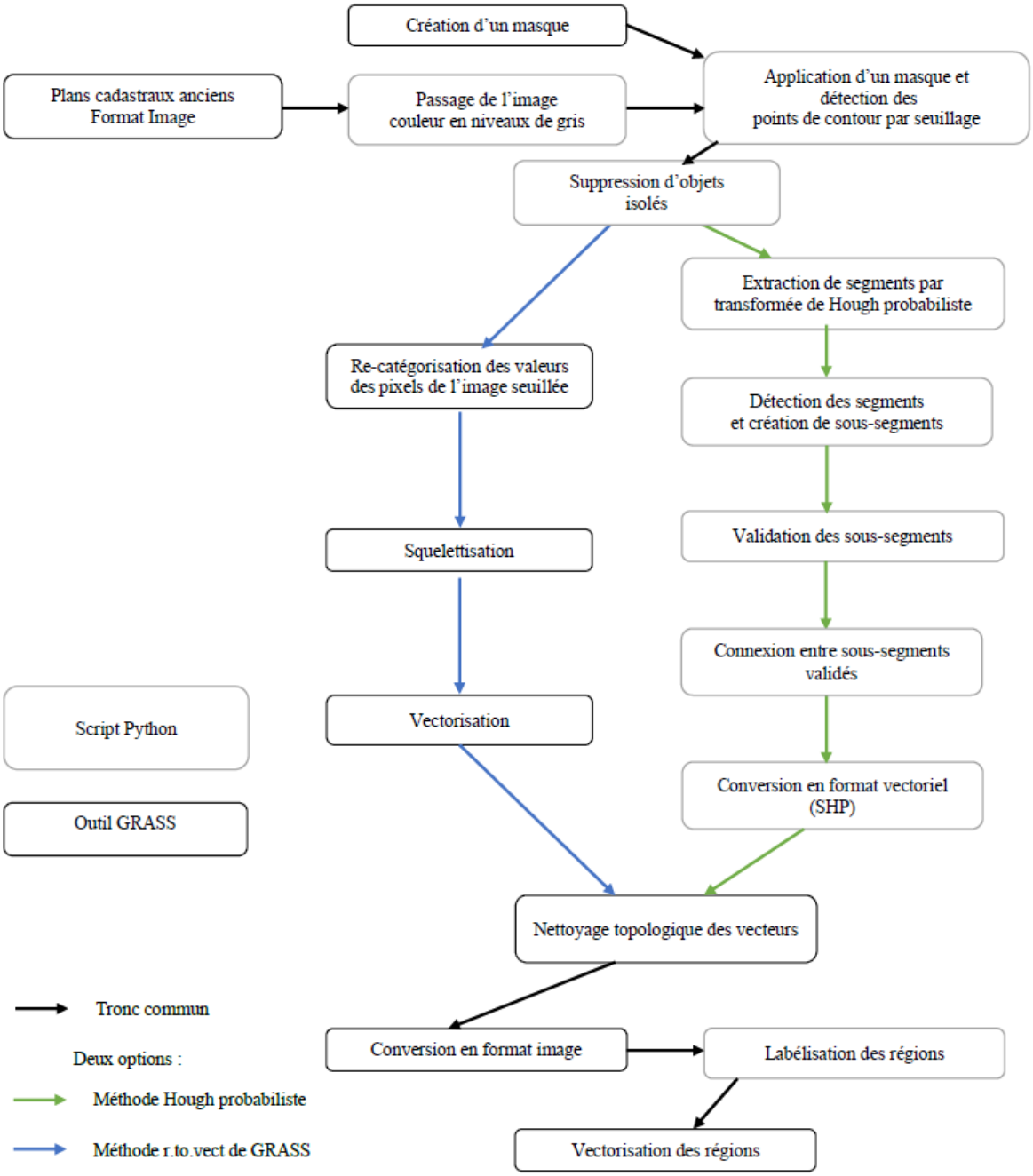
14. Stell JG. Granulation for Graphs. In : Proceedings of the International Conference on Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science, Berlin, Heidelberg, 25 août 1999. Springer-Verlag, p. 417–432.

15. PySal Documentation (Project Jupyter). In : PySal Developers. Python Spatial Analysis Library (PySAL), [en ligne]. Disponible sur : https://pysal.org/documentation. (consulté le 13 mai 2019)

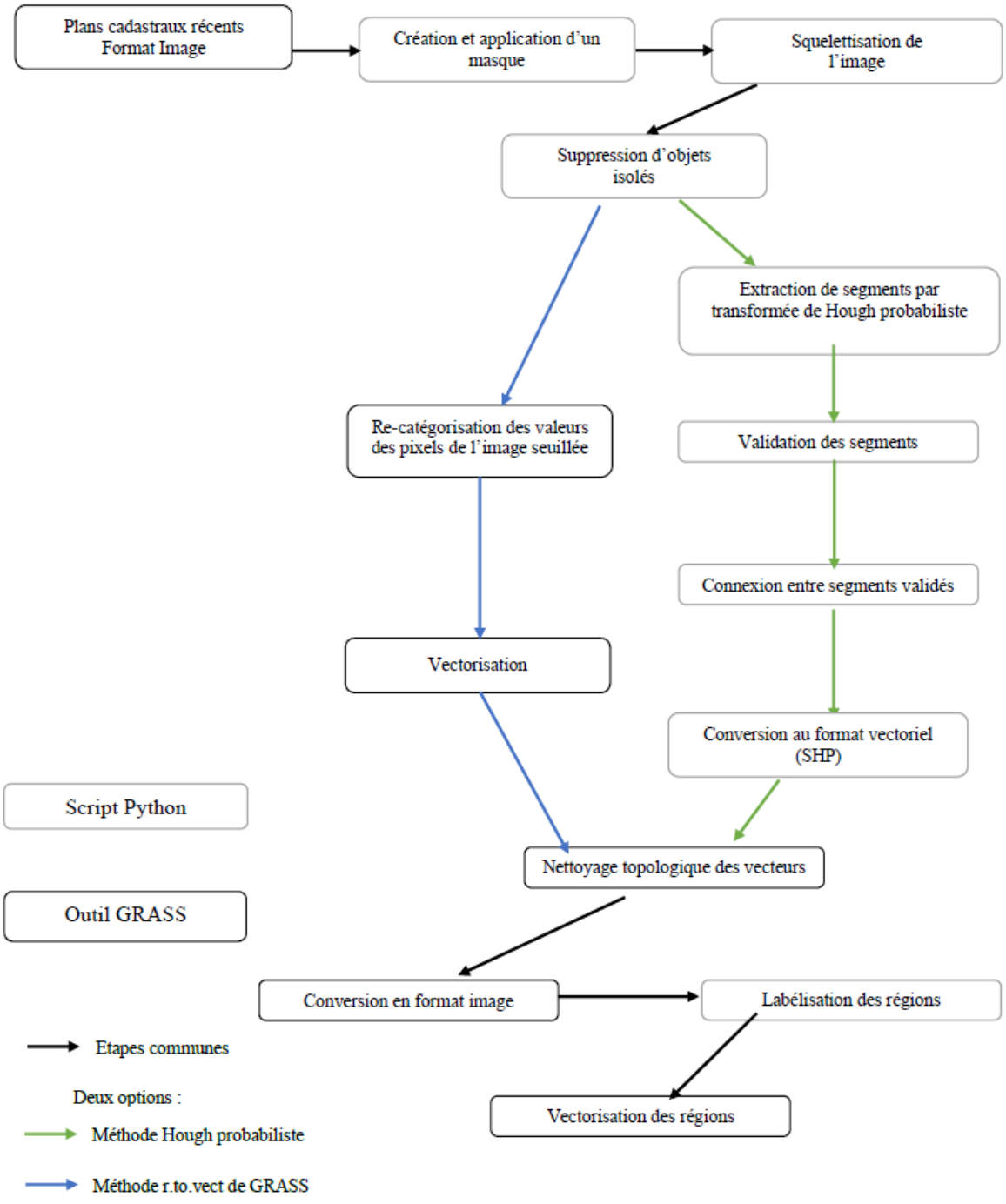
16. Creating a GeoDataFrame from a DataFrame with coordinates — GeoPandas 0.5.0 documentation. In : GeoPandas developers. GeoPandas, [en ligne]. Disponible sur : https://geopandas.readthedocs.io/en/latest/gallery/create_geopandas_from_pandas.html. (consulté le 13 juin 2019)

Table des annexes

Annexe 1
Étapes de la vectorisation, développées par Charlotte Odie en 2017

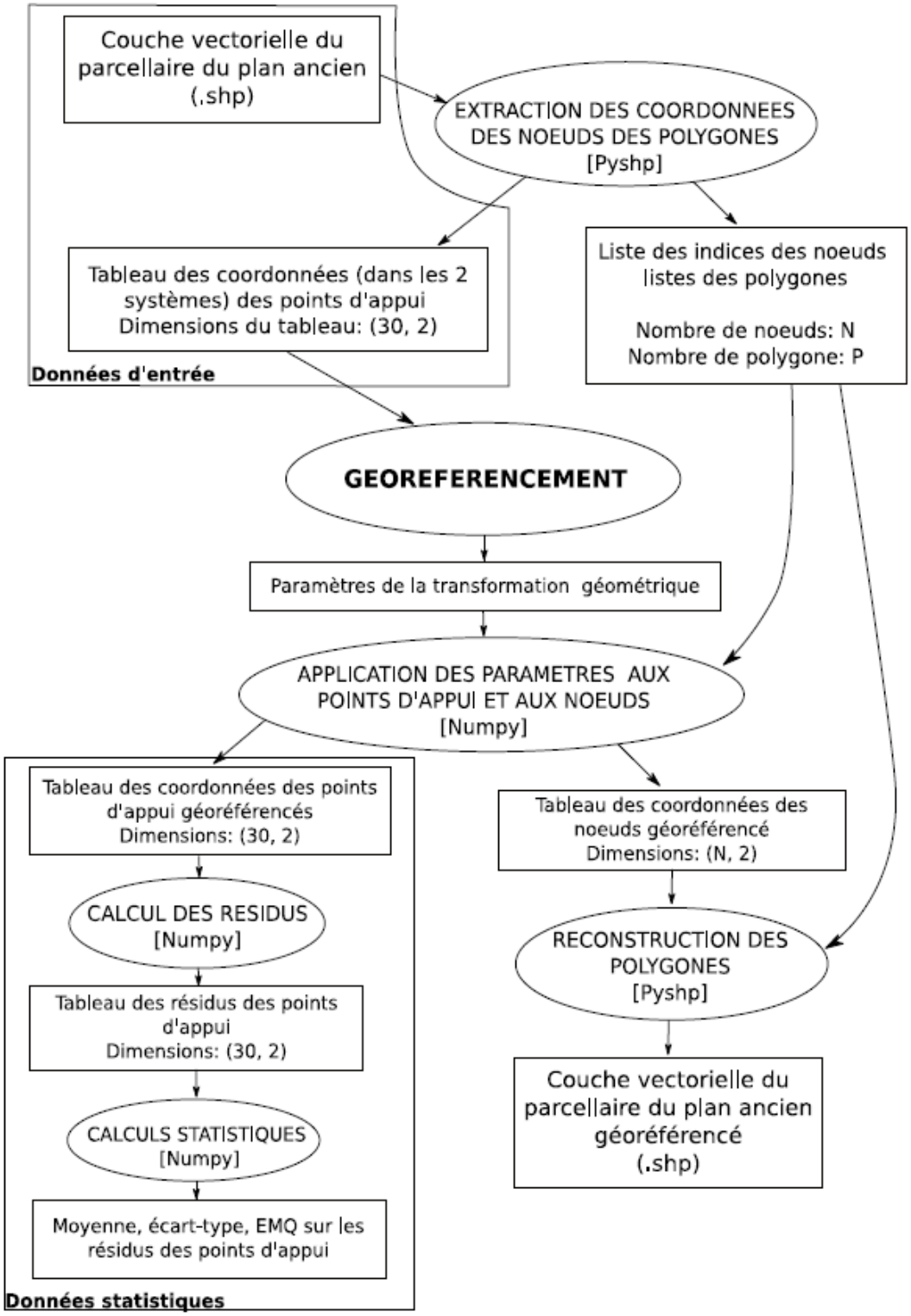


Méthodologie pour les planches anciennes (années 18XX)



Méthodologie pour les planches récentes (années 1970)

Annexe 2
Étapes du géoréférencement, développés par Maïté Fahrasmane en 2016



Annexe 3
Étapes du mosaïquage, développés par Jean-Marc Beveraggi en 2018 (traitement incomplet)

Annexe 4
Détails de la méthode pour exécuter la chaîne de traitement (Mode d'emploi)

Logiciel utile :

Avant de commencer, assurez-vous d'avoir les caractéristiques suivantes :

- La version de Python, supérieure à 3.5. Il n'est pas sûr que la chaîne fonctionne avec une version 3.4, mais cela reste à vérifier.
- Les modules Python suivants doivent être installés : skimage, numpy, matplotlib, Pillow, pysal, pyshp, Shapely, geopandas, Fiona et GDAL.
- Un éditeur de texte de bonne qualité (éviter Wordpad ou Bloc-Notes). Ce n'est pas obligatoire, mais c'est conseillé pour modifier le fichier exe.py et mettre en évidence les éléments importants. Le mieux est d'utiliser un environnement de développement intégré (IDE en anglais), pour récupérer les noms des fonctions du fichier main.py et les paramètres de la fonction principale.

  Par exemple, avec l'IDLE de Python, pour avoir accès à l'ensemble de ces fonctionnalités, il suffit :

  - *De modifier le fichier exe.py en l'ouvrant via l'IDLE*
  - *De l'exécuter une première fois (F5 ou Run > Run Module), puis de l'arrêter dès que le texte « Ouverture du rapport sur l'exécution de la chaîne de traitement » apparaît, SANS FERMER LA CONSOLE (en faisant CTRL+C ou Shell > Interrupt Execution)*
  - *De revenir sur le fichier exe.py, et de le modifier à votre convenance. Lorsque le curseur se trouve dans les parenthèses de la fonction « main », les paramètres s'affichent. Et sur la console, en tapant « dir() », l'ensemble des fonctions disponibles s'affichent, dont celles pour transformer une image en niveaux de gris.*

Les données :

Avant de lancer la chaîne de traitement, il est nécessaire de connaître certains détails sur l'organisation des données dans l'arborescence des dossiers de la chaîne.

Toutes les données initiales doivent être dans le dossier donnees_init. IL N'Y A PAS D'AUTRES POSSIBILITÉS. Pour plus de clarté, il est obligatoire pour chacune de ces données qu'elle soit « rangée » dans une arborescence construite de la façon suivante : *Code INSEE de la commune* / *Année de création de la planche*. Ce cheminement sera ensuite recopié dans le reste des dossiers contenant les calculs à partir de ces planches.

Pour les résultats intermédiaires, qui ne sont pas exploitables dans le cadre des recherches sur l'évolution du territoire, ils sont répartis dans les dossiers de chaque étape, en respectant le cheminement de donnees_init, avec en plus, un dossier correspondant au nom de code de l'exécution (voir le paramètre prefixe) puis un autre dossier correspondant au nom de la planche, lorsque le mosaïquage n'est pas réalisé. Ces informations peuvent être utiles dans le cadre de tests (pour voir le fonctionnement de la chaîne).

Dans le cas d'une interruption prévue de la chaîne pour modification manuelle, les données seront copiées dans donnees_fin / modifs_manuelles directement (sans le cheminement de donnees_init).

Les résultats finaux utiles pour les travaux de recherches seront, eux, dans le dossier donnees_fin, en respectant le cheminement décrit pour les résultats intermédiaires.

Paramétrage de la chaîne de traitement – le fichier exe.py :

Ce qui permet le lancement de la chaîne de traitement est le fichier exe.py. Il recense l'ensemble des paramètres disponibles pour la chaîne, et c'est le seul fichier Python que vous avez le droit de changer. Il est conseillé de modifier le fichier exe.py existant, mais il est possible de créer autant de fichier exe.py (avec un nom différent possible) que souhaité, du moment qu'il respecte la mise en forme suivante :

- Sur la première ligne, doit être écrit « from main import * ». Cette ligne permet de faire appel au fichier main.py, qui exécute la chaîne de traitement.
- Tout commentaire doit être précédé d'un dièse au minimum (ligne qui ne sera pas lu par Python au moment de l'exécution)

- La fonction « main » doit être appelée de la façon suivante (sur une nouvelle ligne) : main (*paramètre 1 = valeur du paramètre 1 souhaitée*, *paramètre 2 = valeur du paramètre 2 souhaitée*, …). Il est possible d'aller à la ligne ou d'ajouter une tabulation autant que vous le souhaitez, sans que cela gêne l'exécution de Python (sauf à l'intérieur d'une chaîne correspondant à un nom ou une valeur de paramètre). Il n'est pas obligatoire de recenser tous les paramètres car il existe une valeur par défaut pour chacun, sauf pour les paramètres « commune », « annee », « planches » et « type_images ».

Il est donc possible de personnaliser le traitement, en modifiant les paramètres. Ceux-ci sont détaillées ci-dessous :

❖ ***Paramètres de base de la chaîne :***

- commune (type str) : Nom du dossier correspondant à une commune (normalement son code INSEE, voire son nom) → 1er niveau de dossier.
- annee (type int ou str) : Nom du dossier correspondant à l'année de la donnée → 2ème niveau de dossier.
- planches (type list) : Liste des noms des planches, sans ajouter le suffixe (correspondant au type de données, qui sera défini dans le paramètre type_images). L'ordre des planches sera l'ordre de traitement, et il devra être le même pour les autres données liées individuellement aux planches, comme les masques et les données de vérité terrain.
- type_images (type str) : Type des images de la planche (faire attention à la casse). Dans notre cas, nous pouvons trouver « JPG » ou « TIF ».
- prefixe (type str) : Nom de code de l'exécution réalisée (utile dans le cas où on exécute plusieurs fois le processus sur un même jeu de données). Ce nom sera celui correspondant au rapport (dans le dossier rapports) et ce sera le 3ème niveau de dossier uniquement dans les résultats intermédiaires et dans donnees_fin. Par défaut, le nom est vide.
- reduction (type list) : Liste contenant les coordonnées pixels que devra étudier la chaîne, dont le format est [[y1,y2],[x1,x2]]. Ainsi, si on souhaite étudier la partie en haut à droite d'une image de taille 512x512, il faudra mettre [[0,200],[350,512]] par exemple. Par défaut, il n'y a aucune réduction.

- fct_NG (type function ou None) : Fonction utilisée pour transformer l'image en niveaux de gris. Trois possibilités : None (si l'image est déjà en niveaux de gris), r2g (fonction rgb2gray de skimage - utile pour une image sur fond blanc) et rgb2gray (dérivée de r2g, mais avec des coefficients différents, pour prendre en compte le fond jauni des planches). Par défaut, la valeur est None.
- max_niveau_print (type int) : Niveau de détail dans l'affichage des informations sur l'exécution de la chaîne (log). Cela permet de limiter le nombre de lignes affichées, pour diminuer le temps et maximiser l'espace sur l'écran. Ces informations, qu'elles s'affichent ou non, sont nécessairement inscrites dans le rapport final. Par défaut, le niveau d'affichage est 2 (doit valoir au minimum 0, et pouvant aller théoriquement jusqu'à l'infini.).
- couleurs_ligne (type list) : Liste de trois nombres entiers, correspondant à la couleur RGB des segments à dessiner sur les images (dans les résultats intermédiaires). Par défaut, c'est [255,0,0], correspondant à la couleur rouge.

❖ ***Paramètres de la vectorisation – le prétraitement***

- v_i1_simpl (type tuple ou None) : Paramètre servant à améliorer la qualité de l'image pour simplifier la détection des segments. Plusieurs possibilités :
  - None : l'image ne sera pas modifiée (sauf pour l'application d'un masque, la binarisation et l'inversion de l'image, si c'est demandé).
  - ('squelettisation', *nbvoisins*) : Réalise une squelettisation de l'image grâce à la fonction skeletonize de skimage. Le paramètre *nbvoisins* correspond au nombre de voisins à utiliser pour la squelettisation (0 pour un voisinage 1x1, 1 pour un voisinage 3x3…).
  - ('hysteresis_local', *nbvoisins*, (*seuil_bas, seuil_haut*), *valeur_finale*) : Modifie une image avec un seuillage par Hystérésis local. *nbvoisins* désigne le nombre de voisins à prendre en compte pour le calcul d'une valeur local (1 pour utiliser les 8 pixels autour, 2 pour utiliser les 24 pixels autour…). *seuil_bas* et *seuil_haut* désigne les facteurs pour évaluer les niveaux des seuils. *valeur_finale* (optionnel) correspond au type de valeur à garder pour l'image seuillée, si le pixel est valide. Ce paramètre peut valoir 'moyenne' pour la moyenne locale, 'maximum' pour le maximum local, ou 'binaire' pour attribuer la valeur 1.

- v_i1_chemin_masque (type list) : Liste des masques à appliquer sur les images servant à la détection des segments (masques dans le même ordre que les planches). ATTENTION : la liste doit être soit vide, soit le nombre d'éléments doit correspondre au nombre de planches. Pour les planches où il n'y a pas de masques, il faut alors mettre None uniquement pour la planche concernée. Par défaut, c'est une liste vide.
- v_i1_inverse (type bool) : Booléen indiquant si l'image servant à la détection des segments doit être inversée pour ce traitement. Si la valeur est True, alors l'image utilisée sera inversée.
- v_i1_binaire (type bool) : Booléen indiquant si l'image servant à la détection des segments doit être binarisée pour ce traitement. Si oui, il y aura en plus la suppression des petits objets, dont la taille est modifiable avec le paramètre v_rso_taille_objet.
- v_i2_simpl : Même paramètre que v_i1_simpl, mais pour l'image servant à la validation des segments (post-traitement). Si la valeur est None, alors l'image utilisée sera la même que pour le traitement.
- v_i2_chemin_masque : Même paramètre que v_i1_chemin_masque, mais pour l'image servant à la validation des segments (post-traitement).
- v_i2_inverse : Même paramètre que v_i1_inverse, mais pour l'image servant à la validation des segments (post-traitement).
- v_i2_binaire : Même paramètre que v_i1_binaire, mais pour l'image servant à la validation des segments (post-traitement).

❖ ***Paramètres de la vectorisation – traitement***

- v_meth (type str ou None) : Nom de la méthode de détection de segments utilisée, parmi 'line_segment_detector' ou 'hough_probabiliste'. Si la vectorisation n'est pas voulue (cela suppose qu'il existe des données avec le chemin correct dans le dossier 1vecto/resultats), alors mettre None. Par défaut, la valeur est None.
- v_lsd_decoup_axe (type int) : Si 'line_segment_detector', c'est le paramètre de découpe selon un des côtés de l'image pour exécuter plusieurs fois l'algorithme LSD. Par exemple, si le paramètre vaut 4, alors LSD sera exécuté 16 fois (image découpée en 4x4 parts égales). Par défaut, le paramètre vaut 1.

- v_lsd_tau (type int ou float) : Si ‘line_segment_detector’, il correspond au paramètre tau de l’algorithme LSD. Par défaut, le paramètre vaut pi/8. Il est très déconseillé de modifier ce paramètre.
- v_hp_threshold (type int) : Si ‘hough_probabiliste’, il correspond au paramètre threshold de la fonction probabilistic_hough_line de skimage. Par défaut, le paramètre vaut 15.
- v_hp_line_length (type int) : Si ‘hough_probabiliste’, il correspond au paramètre line_length de la fonction probabilistic_hough_line de skimage. Par défaut, le paramètre vaut 30.
- v_hp_line_gap (type int) : Si ‘hough_probabiliste’, il correspond au paramètre line_gap de la fonction probabilistic_hough_line de skimage. Par défaut, le paramètre vaut 15.
- v_hp_pas_theta_deg (type float) : Si ‘hough_probabiliste’, il correspond à un pas pour le paramètre theta de la fonction probabilistic_hough_line de skimage. Par défaut, le paramètre vaut 0,5 degré.

❖ ***Paramètres de la vectorisation – post-traitement***

- v_min_intensite (type int ou float) : Valeur de l’intensité d’un niveau de gris pour que le segment auquel le pixel appartient soit validé. Par défaut, sa valeur est 1,0. Il est inutile de le modifier si l’image de validation souhaitée est binaire.
- v_min_pourcent (type tuple) : Tuple indiquant le pourcentage de pixels nécessaire pour qu’un segment soit validé, avec possibilité d’ajouter un critère de longueur du segment. Par exemple, si pour tous les segments, vous voulez avoir 60% de pixels dont le niveau de gris est supérieur à v_min_intensite, alors la valeur du paramètre sera : (0.60,). Si ce pourcentage doit être de 40% pour les segments inférieur à 100 pixels de longueurs, sinon, 60% pour les segments inférieurs à 200 pixels, sinon 20% pour les autres, alors la valeur du paramètre sera : ((100,0.40),(200,0.60),0.20). Par défaut, sa valeur est (0.80,).
- v_epaisseur (type int ou float) : Épaisseur régulière d’une limite de parcelle sur la planche. C’est un paramètre déterminé par l’opérateur, et dont sa détermination doit être la plus juste (au demi-pixel près). Une erreur d’un pixel provoquerait la

suppression quasi-totale des segments, puisque la validation ne laissera pas les segments décalés ; l'erreur sera très vite remarquée. Par défaut, sa valeur est 2,0.

- v_max_long_erreur (type int ou float) : Paramètre (en pixel) permettant au processus de supprimer tous les segments valides dont la longueur est inférieure à celle-ci, lors de l'utilisation de Shapely. Par défaut, sa valeur est 20.
- v_prec_finale (type int) : Précision en pixel des segments validés. Paramètre utilisé pour dilater l'image de validation et pour les statistiques finales de bonnes détections et de fausses alarmes à partir d'images vérité terrain. Par exemple, si le paramètre vaut 2, alors le segment peut être validé si la distance maximum entre un des pixels valides et le segment est de 2 pixels. Par défaut, sa valeur est de 1 pixel.
- v_correct_auto (type bool) : Paramètre permettant d'arrêter automatiquement la chaîne de traitement pour réaliser des modifications manuelles sur l'ensemble des parcelles détectées (le paramètre vaut False dans ce cas). L'arrêt devra provoquer la copie des fichiers utiles dans le dossier donnees_fin / modifs_manuelles. Lorsque les modifications sont terminées, il faut appuyer sur la touche Entrée pour que la chaîne mette à jour les données, les suppriment de donnees_fin et le processus se poursuit. Par défaut, la valeur est True (pas d'arrêt de la chaîne). Attention, cette étape n'est pas encore mise en place.
- v_chemin_compar_lim (type list) : Liste des images de vérité terrain des limites de parcelles, servant à l'évaluation de la qualité de la vectorisation. La liste doit être paramétrée de la même façon que pour les masques. Sa valeur par défaut est une liste vide.
- v_chemin_compar_parc (type list) : Liste de fichiers vectoriels de vérité terrain des parcelles, servant à l'évaluation de la qualité de la vectorisation. La liste doit être paramétrée de la même façon que pour les masques. Sa valeur par défaut est une liste vide.

❖ ***Paramètres du géoréférencement et du mosaïquage***

- g_meth (type str ou None) : Nom de la méthode de géoréférencement utilisée. Une méthode seulement est implémentée (car la plus pertinente selon Maïté Fahrasmane ; voir son mémoire de TFE de 2016) : 'Régression ridge à noyau gaussien'. Le paramètre peut aussi être égal à None, si le géoréférencement n'est pas souhaité. Dans ce cas, pour que la chaîne de traitement puisse se poursuivre, il faut que des résultats existent dans le bon cheminement de dossier (comme pour v_meth).
- m_meth (type bool) : Si True, alors l'opération de mosaïquage se réalisera, sauf si le nombre de planches étudiées est inférieure à 2. Par défaut, la valeur est False.
- m_graphe_adjacence (type bool) : Si True, le graphe d'adjacence sera déterminé et utilisé pour améliorer le mosaïquage. Attention : en raison de bugs sur les parcelles mosaïquées, il est encore impossible de réaliser cette étape. Aucune vérification n'a été réalisée sur son fonctionnement.
- m_distance_densif (type int) : Distance maximale autorisée entre 2 points situées aux frontières des ilots de parcelles. La densification permet ensuite de réaliser une triangulation de Delaunay, utile pour déterminer les relations d'adjacence. Paramètre non testé.
- chemin_granulaire (type str) : Chemin d'accès au graphe granualaire. Paramètre non utilisé.
- chemin_temporel (type str) : Chemin d'accès au graphe temporel. Paramètre non utilisé.

Annexe 5
Considérer des segments parallèles selon une certaine précision

Mathématiquement, il existe plusieurs façons de déterminer si deux segments sont parallèles, comme le produit vectoriel nul ou l'égalité des coefficients directeurs. Mais aucune des formules ne permettent d'y introduire une imperfection, qui soit de plus associée à un paramètre de longueur. Par exemple, nous pourrions penser qu'il suffirait d'insérer une différence acceptable entre deux coefficients directeurs pour considérer deux droites parallèles, mais selon si les droites sont quasi horizontales ou verticales, la différence calculée n'est pas la même.

Démonstration :

Soit deux droites dont l'angle varie de 0.01 rad, que nous considérons parallèles. Dans le cas de quasi-horizontalité, si la première droite a comme coefficient directeur 0.01, alors l'autre droite aura comme coefficient directeur 0.00 ou 0.02, donc une différence de 0.01. Dans le cas de quasi-verticalité, si la première droite a comme coefficient directeur 100, l'autre droite aura comme coefficient directeur 49.99 ou -3 000 180.04, ce qui est incomparable.

Nous avons donc dû définir le terme de droites « pseudo-parallèles » selon une certaine précision p (scalaire positif). Cette expression est définie grâce aux deux propriétés et la définition suivantes

Propriété 1 : Si deux droites ne s'intersectent pas, alors elles sont pseudo-parallèles selon n'importe quelle précision p.

Propriété 2 : Soient deux droites s'intersectant en un point C. Soit un cercle de rayon p (nombre réel positif en pixel) et de centre C. Deux droites sont pseudo-parallèles à une précision p si deux des quatre points à l'intersection entre les droites et le cercle, sont éloignés d'une distance inférieure ou égale à 0.02 pixel (valeur arbitraire). Exemple dans la Figure 30.

Définition : Deux droites sont pseudo-parallèles à une précision p et si seulement si la propriété 1 ou la propriété 2 est vérifiée.

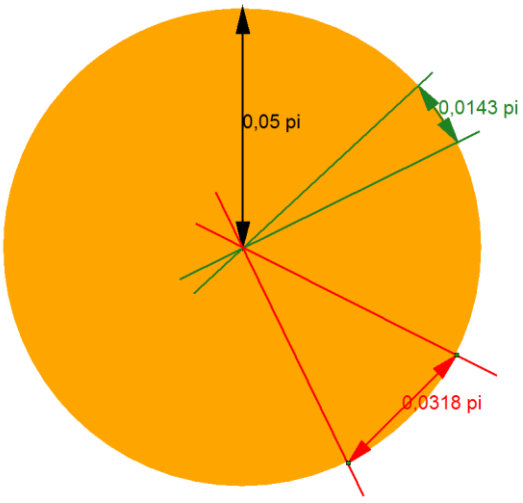


Figure 30 : Le pseudo-parallèlisme - en vert, deux droites pseudo-parallèles selon la précision de 0.05 pi, et en rouge, deux droites non pseudo-parallèles selon la précision de 0.05 pi. Ici, pi signifie pixel.

Annexe 6
Liste des différents processus possibles

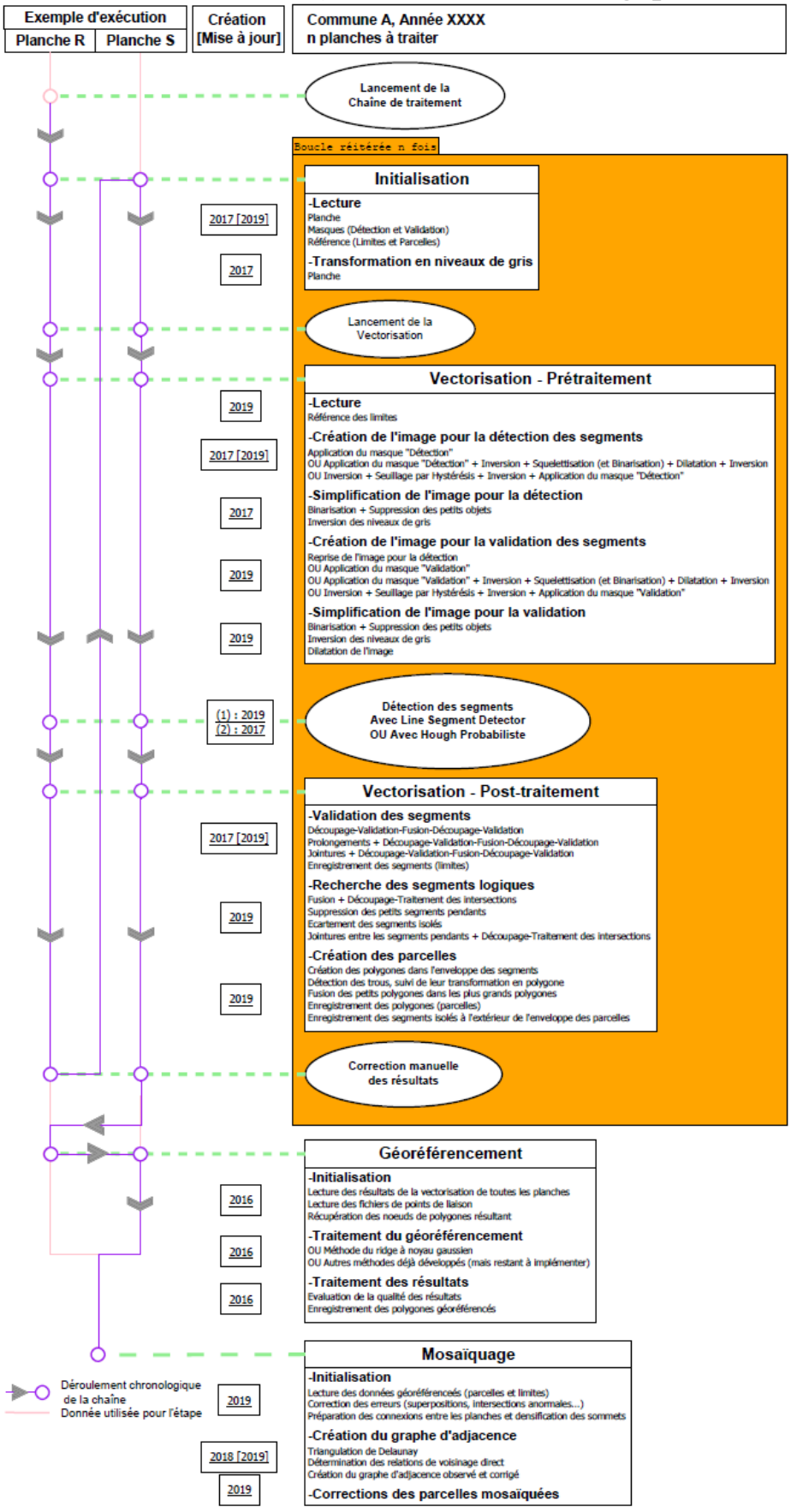
Proc.	PGTAM	PAMGT	PMGT	PGAM (1)	PAMG (1)	PMG (1)	PGAM (2)	PAMG (2)	PMG (2)
Données	Données autre(s) année(s)	Données autre(s) année(s)	Données autre(s) année(s)				Données autre(s) année(s)	Données autre(s) année(s)	Données autre(s) année(s)
							Centre des parcelles	Centre des parcelles	Centre des parcelles
							Graphe temporel	Graphe temporel	Graphe temporel
	Planche	Planche	Planche	Planche	Planche	Planche	Planche	Planche	Planche
	Tableau d'assemblage	Tableau d'assemblage		Tableau d'assemblage	Tableau d'assemblage		Tableau d'assemblage	Tableau d'assemblage	
	Graphe granulaire	Graphe granulaire		Graphe granulaire	Graphe granulaire		Graphe granulaire	Graphe granulaire	
	Points de liaison	Points de liaison	Points de liaison	Points de liaison	Points de liaison	Points de liaison	Points de liaison	Points de liaison	Points de liaison
Étapes immuables	Détection des limites	Détection des limites	Détection des limites	Détection des limites	Détection des limites	Détection des limites	Détection des limites	Détection des limites	Détection des limites
	Détection des caractères	Détection des caractères	Détection des caractères	Détection des caractères	Détection des caractères	Détection des caractères			
Étapes perfectibles	Détection des parcelles	Détection des parcelles	Détection des parcelles	Détection des parcelles	Détection des parcelles	Détection des parcelles	Détection des parcelles	Détection des parcelles	Détection des parcelles
	Géoréférencement			Géoréférencement			Géoréférencement		
	Graphe temporel								
	Graphe d'adjacence	Graphe d'adjacence		Graphe d'adjacence	Graphe d'adjacence		Graphe d'adjacence	Graphe d'adjacence	
	Mosaïquage	Mosaïquage	Mosaïquage	Mosaïquage	Mosaïquage	Mosaïquage	Mosaïquage	Mosaïquage	Mosaïquage
		Géoréférencement	Géoréférencement		Géoréférencement	Géoréférencement		Géoréférencement	Géoréférencement
		Graphe temporel	Graphe temporel						

Les codes donnés aux processus correspond à l'ordre d'exécution des étapes perfectibles. Les textes en couleurs dans les données et les étapes immuables informent que ce sont des options, qui ne modifie pas le processus (les options d'une même couleur indiquent qu'elles doivent être présentes en même temps). Les scénarios avec une case orange en haut sont des scénarios impossibles à réaliser, car pour que le mosaïquage se réalise avant, il faut nécessairement avoir le graphe granulaire en donnée initiale.

Processus PAMGT (col. 2) : Deux scénarios possibles pour ce processus. Les données initiales sont la planche à étudier, un fichier de points de liaisons, le graphe granulaire réalisé manuellement à partir du tableau d'assemblage, et les données issues d'une exécution précédente de la chaîne de traitement sur une même zone géographique mais à une époque différente. Une étape immuable en option est la détection de caractères, qui améliorera les résultats des étapes perfectibles sans modifier leur ordre d'exécution. Les étapes perfectibles se réalisent dans cet ordre : Détection des parcelles (grâce aux limites et éventuellement aux caractères), le graphe d'adjacence (grâce aux parcelles détectées et au graphe granulaire), le mosaïquage (grâce au graphe d'adjacence, au graphe granulaire et les parcelles détectées), le géoréférencement (grâce aux parcelles détectées et aux poins de liaison) le graphe temporel (grâce aux parcelles géoréférencées et au graphe d'adjacence et les données des autres années).

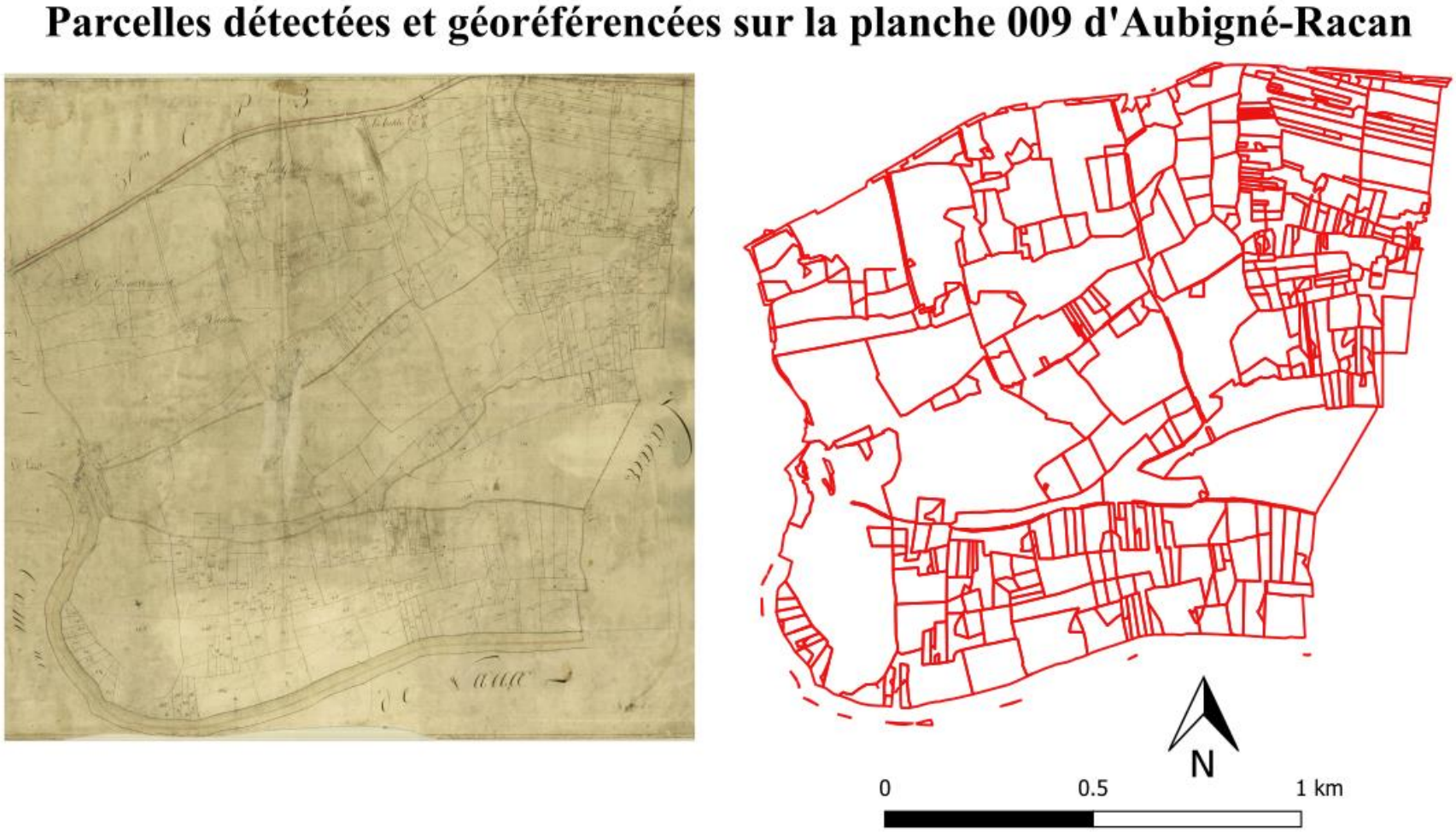
Processus PGAM (2) (col. 7) : Quatre scénarios possibles pour ce processus. Les données initiales sont la planche à étudier, un fichier de points de liaisons et les centres des parcelles (couche manuelle). La détection de caractères n'est pas réalisée puisque l'information recherchée par cette détection est présente dans la couche des centres de parcelles. On peut ajouter les les données issues d'une exécution précédente de la chaîne de traitement sur une même zone géographique mais à une époque différente, ce qui oblige la création manuelle du graphe temporel (puisque nous avons aussi les centres de parcelles ; couches réalisées simultanément). On peut aussi ajouter le tableau d'assemblage, ce qui nécessite de réaliser manuellement le graphe granulaire. Ces dernières données ne modifient pas l'ordre d'exécution des étapes perfectibles, mais elles en améliorent les résultats. Les étapes perfectibles sont réalisées dans cet ordre : Détection des parcelles (grâce aux limites détectées, les centres de parcelles et éventuellement au graphe temporel), géoréférencement (grâce aux parcelles détectées et aux points de liaison), graphe d'adjacence (grâce aux parcelles géoréférencées, aux centres de parcelles et éventuellement au graphe temporel) et mosaïquage (grâce au graphe d'adjacence, aux parcelles géoréférencées et éventuellement au graphe granulaire).

Annexe 7
Chaîne de traitement actuel (dans l'ordre chronologique d'exécution)

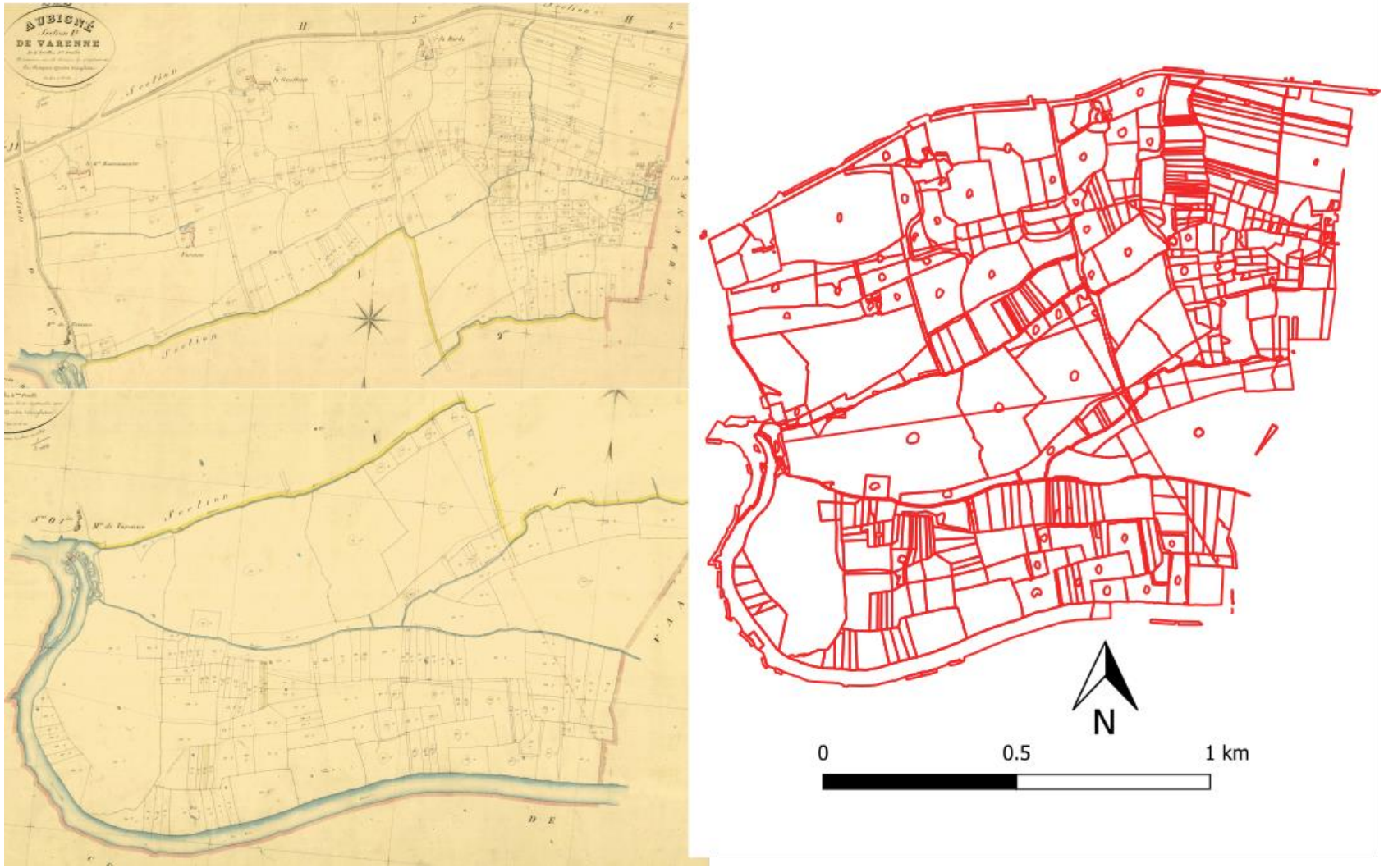


Annexe 8

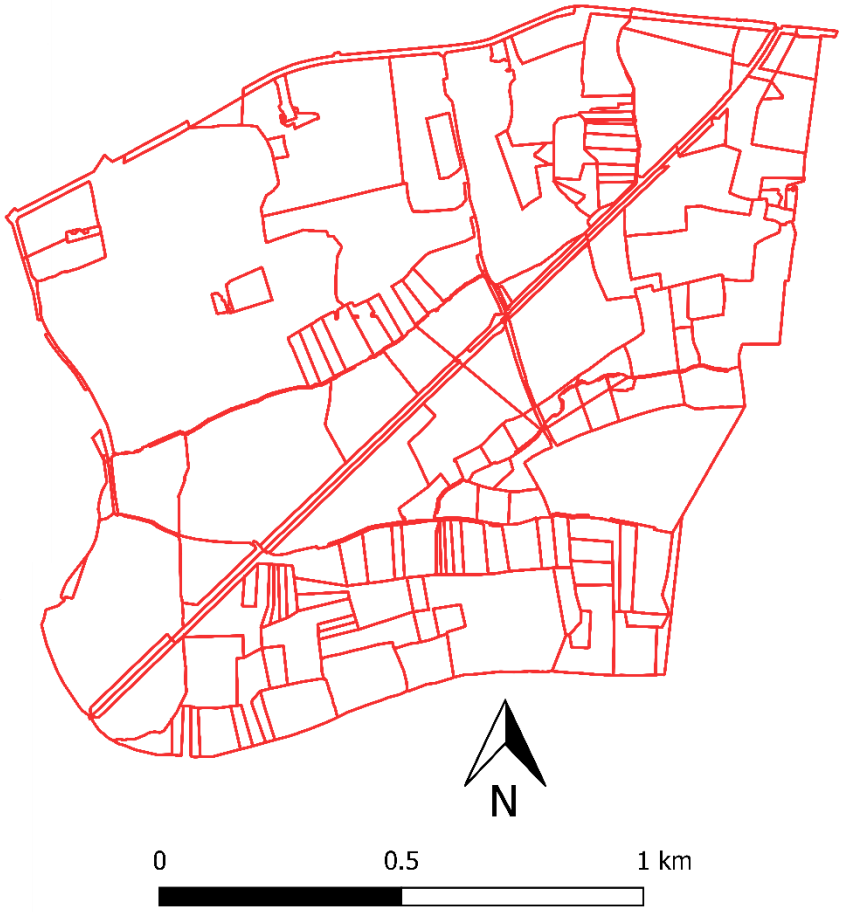
Résultats à l'issue de l'exécution complète de la chaîne de traitement (d'abord la planche 009 de 1813, puis les planches 114 et 115 de 1850, et les planches 151 et 152 de 1972, pour la commune d'Aubigné-Racan)



Parcelles détectées, géoréférencées et mosaïquées sur les planches 114 et 115 d'Aubigné-Racan



Parcelles détectées, géoréférencées et mosaïquées sur les planches 151 et 152 d'Aubigné-Racan



Liste des figures

Liste des tableaux

Vectorisation du cadastre ancien : restructuration de la chaîne de traitement, implémentation d'une nouvelle méthode de détection et utilisation de la théorie des graphes.

Mémoire d'Ingénieur C.N.A.M., Le Mans 2019

RESUME

Depuis sa création dans les années 1800, le cadastre français n'a cessé de décrire la forme des territoires. Dans l'objectif de faciliter l'étude de l'évolution des territoires par les planches cadastrales, le Laboratoire Géomatique et Foncier réalise depuis 2016 une chaîne de traitement semi-automatisée pour vectoriser, géoréférencer et mosaïquer ces planches.

Dans ce TFE, nous nous sommes concentrés sur la mise à jour du code de la chaîne de traitement dont les principales étapes avaient été déjà programmées. Une révision totale sera réalisée, l'étape cruciale de vectorisation sera retouchée pour y implémenter une nouvelle méthode, le *Line Segment Detector*, et la chaîne finale sera imaginée grâce à la théorie des graphes. L'objectif ultime, quasiment atteint à la fin de ce TFE, est l'utilisation unique de Python pour l'exécution complète de la chaîne.

Mots clés : Cadastre, Vectorisation, *Line Segment Detector*, Théorie des Graphes, Python, SIG, Shapely, Chaîne de traitement.

SUMMARY

Since its creation in the 1800s, the French Land registry has never stopped describing the shape of the land settlements. In order to facilitate the study of the land settlement evolution by the cadastral plate, the Laboratoire Géomatique et Foncier has been producing a semi-automatic numerical processing chain since 2016 to vectorizing, geocoding and tiling these plates.

In this TFE, we focused on updating the numerical processing chain's code whose main steps had already been programmed. We will carry out a complete overhaul, the crucial stage of vectorization will be altered to implement a new algorithm, the Line Segment Detector, and the final numerical processing chain will be redesigned through the Graph Theory. The ultimate goal, which was almost achieved at the end of this TFE, is the sole use of Python for the complete running of the numerical processing chain.

Key words : Cadastre, Vectorization, Line Segment Detector, Graph Theory, Python, GIS, Shapely, Numerical processing chain.