Optimisation de la chaîne de production de plans d'intérieur depuis des nuages de points 3D

Damien Richard

▶ **To cite this version:**

Damien Richard. Optimisation de la chaîne de production de plans d'intérieur depuis des nuages de points 3D. Sciences de l'ingénieur [physics]. 2024. dumas-04842634

HAL Id: dumas-04842634

https://dumas.ccsd.cnrs.fr/dumas-04842634v1

Submitted on 17 Dec 2024

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

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

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

ÉCOLE SUPÉRIEURE D'INGÉNIEURS GÉOMETRÈS ET TOPOGRAPHES

MÉMOIRE

Présenté en vue d'obtenir

Le DIPLOME D'INGÉNIEUR CNAM

SPÉCIALITÉ : Géomètre et Topographe

par

Damien RICHARD

Optimisation de la chaîne de production de plans d'intérieur depuis des nuages de points 3D

Soutenu le 02 septembre 2024

JURY

SIMONETTO Élisabeth	Président du jury
FRENEAT Gael	Maître de stage
CHARLET Christophe	Enseignant référent

Remerciements

Je souhaite tout d'abord exprimer ma gratitude envers l'ensemble du personnel du cabinet ATRIUM Foncier pour leur accueil et leur soutien tout au long de ce projet de fin d'études.

Je remercie particulièrement Monsieur Gael Frénéat, géomètre-expert, pour la confiance qu'il m'a accordée dans la réalisation de ce projet, ainsi que pour ses conseils avisés tout au long de ce travail.

Je tiens également à remercier Monsieur Christophe Charlet, mon enseignant référent, pour sa disponibilité et ses précieux conseils qui ont grandement contribué à l'avancement de ce stage.

Liste des abréviations :

2D : Deux dimensions (axe X, axes-Y)

3D : Trois dimensions (axe X, axes-Y, axe Z)

API : Application programming interface, interface de programmation d'application

CAO : Conception assisté par ordinateur

CPU : Central processing unit (Processeur)

DAO : Dessin assisté par ordinateur

GNSS : Global Navigation Satellite System

GPU : Graphical processing unit (Carte Graphique)

LIDAR : Light Detection And Ranging

NRTK : Network Real Time Kinematic

RAM : Random access memory (Mémoire vive)

SAM : Segment anything Model

VRAM : Video random access memory (Mémoire vive vidéo)

Glossaire

Le Dessin Assisté par Ordinateur (DAO) : est la discipline qui consiste à produire des dessins techniques, des schémas, des montages, des ébauches, des plans de fabrication… en utilisant différents logiciels informatiques. En DAO, le crayon et les autres instruments de dessin font place à la souris et au clavier.

Dataset/Jeu de données : est une collection de données structurées associées à un domaine, une thématique ou un secteur d'activité spécifique. Les datasets peuvent contenir différents types d'informations, comme des chiffres, du texte, des images, des vidéos et même des fichiers audios. Ils peuvent se présenter sous divers formats comme CSV, JSON ou SQL.

Deep Learning (Apprentissage Profond) : Branche du Machine Learning utilisant des réseaux de neurones profonds pour modéliser des abstractions complexes dans les données. Efficace pour la reconnaissance d'images, le traitement du langage naturel et la reconnaissance vocale.

Feature Backbone : est un terme utilisé en apprentissage automatique, pour désigner une architecture de réseau de neurones (généralement un réseau convolutif) qui est utilisée pour extraire des caractéristiques (features) pertinentes à partir d'entrées brutes, telles que des images.

Intelligence Artificielle (IA) : Domaine de l'informatique visant à créer des systèmes capables de réaliser des tâches nécessitant l'intelligence humaine, comme la compréhension du langage, la reconnaissance des formes et la prise de décision.

Interface de programmation d'application : est un ensemble de définitions et de protocoles qui facilite la création et l'intégration des applications.

Machine Learning (Apprentissage Automatique) : sous-discipline de l'intelligence artificielle développant des algorithmes permettant aux ordinateurs d'apprendre à partir de données et de faire des prédictions ou des décisions sans être explicitement programmés.

Modèle Transformer : est une architecture de réseau de neurones qui utilise des mécanismes d'attention pour traiter des séquences de données en parallèle, plutôt que de manière séquentielle.

NRTK : Technique de positionnement par satellite basée sur l'utilisation de mesures de phase des ondes porteuses des signaux émis par les systèmes GNSS permettant d'obtenir en temps réel des corrections. Ces corrections sont calculées à partir d'un réseau de stations permanentes.

Segmentation : La segmentation d'image est une technique de vision par ordinateur qui divise une image numérique en groupes de pixels distincts (segments d'images) afin de faciliter la détection d'objets et les tâches connexes.

VRAM, Mémoire vidéo : est une forme spécialisée de RAM utilisée dans les unités de traitement graphique (GPU) pour stocker les données d'image pour un écran d'ordinateur. Elle est dédiée uniquement à la gestion des exigences graphiques des applications, telles que les jeux vidéos et les logiciels d'édition vidéo. Il s'intègre directement à la carte graphique et sert de tampon haute vitesse entre le GPU et l'écran.

Widget : Contraction des mots window et gadget pour désigner un élément graphique présent sur un bureau virtuel, permettant d'accéder à un service.

Introduction :

L'essor récent des technologies 3D LIDAR dans les cabinets de géomètres expert a profondément bouleversé les méthodes et habitudes du métier. Ces évolutions facilitent et accélèrent considérablement le travail sur le terrain, tout en complexifiant le post-traitement, remettant ainsi en question l'idée selon laquelle la principale valeur ajoutée du géomètre-topographe réside dans la durée et la complexité de l'intervention sur le terrain.

Traditionnellement, les relevés topographiques classiques, utilisant des instruments tels que les stations totales, les rubans ou les chaînes, impliquent la prise minutieuse de points individuels. Ce processus chronophage rend impossible la collecte de tous les points d'une scène, obligeant l'opérateur à sélectionner un nombre limité mais significatif d'éléments à mesurer. De plus, les données issues de ces relevés se présentent sous forme de listes de points, nécessitant une codification et/ou la réalisation de croquis compréhensibles.

En revanche, l'utilisation des méthodes 3D LIDAR permet des relevés rapides (capturant des milliers de points par minute). L'abondance de données récoltées et la relative ergonomie de ces appareils rendent alors superflu le dessin de croquis et la réflexion terrain précise, sur les éléments à mesurer. Il suffit de positionner l'appareil au bon endroit, d'attendre et de vérifier sur les données 3D visualisables en direct la présence de tous les éléments souhaités. Cependant, si la phase de relevé est nettement plus rapide, le traitement des données est plus long et fastidieux. Le nuage de points généré constitue une donnée volumineuse et difficilement manipulable. En effet, il s'agit d'un ensemble 3D de milliers de points non organisés, à visualiser sur des écrans 2D. De plus, ces données nécessitent une puissance de calcul importante et du personnel qualifié dans ce domaine spécifique et relativement nouveau. Le gros du travail du géomètre se déplaçant alors de l'intervention terrain vers le traitement bureau.

Les cabinets de géomètres-experts sont ainsi confrontés à une réduction significative du temps et de la complexité des relevés topographiques, mais également à une augmentation des délais et des difficultés de traitement des mesures. Face à ce besoin d'optimisation et de simplification des processus de traitement des nuages de points 3D, le cabinet de géomètre-expert ATRIUM Foncier, m'a chargé d'explorer et de mettre en place des solutions pour améliorer et automatiser les traitements.

Parmi les étapes de production (géoréférencement, recalage des nuages de points, nettoyage des nuages), il a été décidé de se concentrer sur l'automatisation vectorisation des nuages de points 3D, en somme du dessin des plan 2D (étape dont l'automatisation est la plus utile et réalisable).

De plus face à la complexité supérieure des nuages de points d'extérieur, et au fait que ces types de relevés sont moins fréquemment réalisés par scanner LIDAR terrestre, nous nous sommes concentrés sur le traitement unique des nuages de points d'intérieur. Nous souhaitons donc détecter/dessiner les éléments essentiels d'un plan d'intérieur, tels que les murs, les ouvertures, les hauteurs sous plafond et les surfaces des pièces.

Il convient alors, pour répondre à ce défi, d'examiner d'abord l'état de l'art, ici nous présenterons uniquement les solutions de segmentation et classification d'images (les raisons seront évoquées dans ce mémoire). Après avoir fait l'inventaire des solutions et méthodes existantes, nous nous attarderons sur les procédures et techniques de relevé terrain, ainsi que de traitements utilisés par le cabinet, et ce, afin d'identifier les contraintes de notre contexte de travail, et de définir les spécifications des outils à exploiter et à développer, en somme afin d'établir un 'cahier des charges'. Une fois les solutions existantes connues et les contraintes comprises, nous expliquerons les deux méthodologies développées. Enfin, nous présenterons et comparerons, en détail, les résultats, des deux solutions, afin de déterminer la pertinence et la juteuse des méthodes présentées.

I État de l'art et contexte de l'étude

Dans cette partie, nous allons, présenter les méthodes existantes qui pourront nous aider dans notre tâche. Nous nous intéresserons ensuite aux caractéristiques de la chaîne de travail du cabinet ATRIUM Foncier, allant de la prise de mesures au dessin du plan. Ceci nous permettra d'établir les contraintes et opportunités auxquelles nous serons confrontés lors du développement de nos solutions d'automatisation du dessin de plans d'intérieur à partir de nuages de points 3D.

I.1 État de l'art

Les solutions d'automatisation du dessin de plan d'intérieur développés notamment au cours des deux années précédentes à l'ESGT, Talec 2017 et Kobena 2023, reposent essentiellement sur des méthodes de calcul et traitement d'images/nuages de points 'classiques' (transformés de Hough triangulation de Delaunay, RANSAC ….), Or, les techniques d'intelligence artificielle (deep/machine learning), pourraient s'avérer très utiles pour cette tâche, du fait de leurs compréhensions holistiques des données d'entrée. Et, justement, les avancées récentes dans le secteur du machine/deep Learning furent fulgurantes. Nous avons alors pensé qu'il serait intéressant d'apporter une solution fondée sur de telles méthodes et, allons, donc, dans cette partie examinée les caractéristiques des modèles et leurs entrainements.

I.1.1 Segmentation d'image, le Segment Anything model (SAM)

Nous explorons d'abord les possibilités de segmentation d'image 2D générale, qui ne sont pas spécifiquement liées à la reconstruction de plans d'intérieur. Dans ce domaine, le Segment Anything Model (SAM), Kirillov 2023, a retenu notre attention. Il s'agit en effet d'un modèle de segmentation d'image, de pointe publiée en open-source par Meta et doté d'une API Python.

Atout majeur de SAM, il possède une capacité de généralisation zéro-shot exceptionnelle (Figure n°1, ci-dessous). Pour arriver à ce résultat SAM a été entraîné sur un grand volume de données d'entraînement (il s'agit du dataset/jeu de données Sa-1B) pour comprendre et segmenter une grande variété d'images. Cela lui permet de généraliser ses

prédictions à une multitude d'objets et de scènes, même s'il ne les a jamais vu auparavant, sans nécessiter d'entraînement supplémentaire.

Concrètement, SAM est un système de segmentation qui fonctionne avec une invite de commande, qui peut être un pointage ou une boîte englobante. Il peut "découper" n'importe quels objets ou éléments, dans n'importe quelles images. Cela signifie que l'on peut utiliser SAM pour segmenter des objets spécifiques dans une image, simplement en dessinant une boîte autour de l'objet ou en le pointant. Afin de mieux saisir le fonctionnement du modèle, voici ci-dessous (figure n°2) un diagramme détaillant son fonctionnement. Mais nous pouvons succinctement expliquer, que, le Segment Anything Model (SAM) encode une image en entrée pour extraire ses caractéristiques visuelles essentielles. Simultanément, des invites (points, boîtes) sont encodés pour spécifier les zones d'intérêt. Les représentations d'image et les encodages des invites sont combinés et traités par un décodeur de masque pour générer plusieurs masques de segmentations possibles. Chaque masque est associé à un score de confiance, indiquant sa pertinence par rapport aux prompts fournis, permettant ainsi de segmenter précisément les objets dans l'image. SAM génère des masques de sortie qui sont des images binaires, masquant le ou les objets d'intérêt dans l'image.

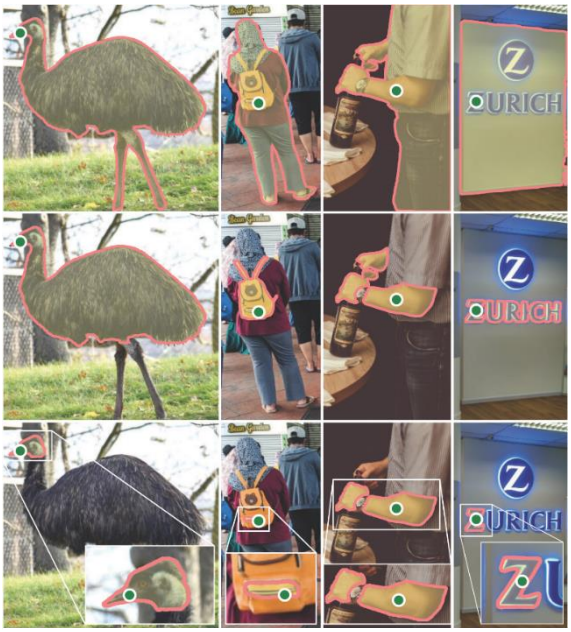


Figure 1: Segmentation SAM (Source : https://github.com/facebookresearch/segment-anything)

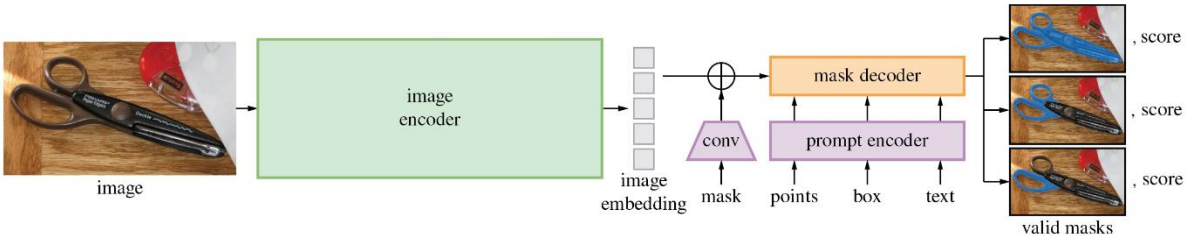


Figure 2 : Diagramme SAM (Source : https://github.com/facebookresearch/segment-anything)

En somme, SAM offre des capacités de segmentation d'images sans précédent. Il pourrait ainsi être utilisé pour segmenter, et donc vectoriser, les représentations 2D (ortho-images, cartes de densité, etc.) des nuages de points 3D.

I.1.2 Reconstruction des formes depuis une carte de densité grâce au modèle RoomFormer

Abordons désormais les méthodes de vectorisation spécifiques à la reconstruction de plans d'intérieur. Dans ce domaine précis, il existe peu d'études, cependant, une s'est avérée être particulièrement pertinente, il s'agit de l'article de Yue 2023, « Floorplan Reconstruction Using Two-Level Queries. ». En effet, la méthode présentée, RoomFormer, est un modèle Transformer pour la reconstruction de plans d'étage en une seule étape. Il a été présenté lors de la conférence CVPR 2023 (Computer Vision and Pattern Recognition Conference).

Pour générer des plans d'intérieur précis et détaillés en une seule étape, RoomFormer utilise une approche de requêtes à deux niveaux pour générer les polygones de plusieurs pièces en parallèle, de manière holistique, sans étape intermédiaire manuelle. Concrètement, le fonctionnement de RoomFormer est détaillé dans le diagramme ci-dessous (Figure n°3) :

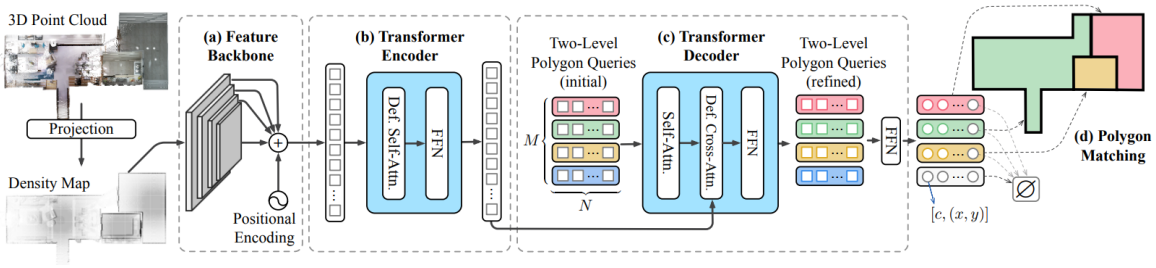


Figure 3 : Diagramme RoomFormer (Source : Floorplan Reconstruction Using Two-Level Queries, Yue 2023)

Ainsi le modèle RoomFormer prend en entrée une carte de densité issue de la projection d'un nuage de points 3D d'intérieur. Cette carte est ensuite traitée par un réseau de caractéristiques (Feature Backbone) pour extraire des caractéristiques visuelles importantes. Ces caractéristiques sont encodées par un encodeur de Transformer qui utilise des mécanismes d'attention (Les mécanismes d'attention permettent aux modèles de Transformer de se concentrer sur les parties essentielles des données d'entrée en calculant des poids d'importance pour chaque élément. Ils aident à capturer les dépendances distantes et à générer des représentations contextuelles précises.) et des réseaux d'anticipations (ce sont des réseaux de neurones à couches entièrement connectées qui appliquent des transformations non linéaires aux données, améliorant ainsi la capacité des modèles à capturer des relations complexes et des patterns, dans les modèles de Transformer, ils suivent les mécanismes d'attention pour enrichir les représentations des données). Des requêtes de polygones à deux niveaux sont initialisées pour représenter les pièces, puis affinées par un décodeur, Transformer, utilisant des

mécanismes d'attention. Les requêtes affinées passent par un réseau pyramidal de caractéristiques, FPN, qui est une architecture de réseau de neurones utilisée en vision par ordinateur pour détecter des objets à différentes échelles. Il fusionne des caractéristiques à plusieurs résolutions, permettant au modèle de conserver des détails fins tout en capturant le contexte global, ce qui améliore la précision et la robustesse de la détection et de la segmentation des objets, assurant la cohérence à différents niveaux de détail. Enfin, les polygones sont appariés et validés pour correspondre précisément aux pièces dans l'image d'origine, résultant en une représentation détaillée et précise de l'espace intérieur. Nous le verrons plus amplement en partie II.1.2, nous avons repris la projection 2D de nuage de points 3D sous forme de carte de densité.

Enfin, l'implémentation Python de la méthode, ainsi que les fichiers de poids du modèle pré-entraîné sur les jeux de données SceneCAD et Structure 3D, ont été mis à disposition (répertoire GitHub). De plus, RoomFormer offre des temps d'inférence bien plus rapides et des résultats bien plus précis que les méthodes précédentes (MonteFloor, HEAT …). En outre, on constate, que RoomFormer offre une solution inédite et apparemment idéale à notre problème de vectorisation automatisée de nuages de points 3D.

I.2 Contexte de l'étude

Maintenant que nous connaissons les méthodes et solutions actuelles, nous nous attarderons sur le contexte dans lequel s'inscrit notre travail. Plus concrètement, nous allons ici détailler les objectifs de notre étude puis, les processus, méthodes et manipulations d'usage dans le cabinet de géomètre expert ATRIUM Foncier, de la prise des mesures aux dessins des plans, et ceux en passant par les divers traitements appliqués aux nuages de points.

I.2.1 Objectifs de l'étude

Nous avons précédemment indiqué que nous nous sommes concentrés sur l'automatisation du processus de production des plans d'intérieur. Il convient alors d'établir la liste des éléments que nous souhaitons traiter/dessiner/calculer, automatiquement.

Nous avons ainsi défini la liste suivante (élément à détecter et dessiner/afficher), par ordre de priorité :

- Murs (limites des pièces)
- Hauteurs sous plafond ou faux plafond
- Altitudes (niveaux du sol)

Ces éléments ont été choisis, car ils forment ensemble l'essentiel du contenu d'un plan d'intérieur (en quelque sorte le fond, le sens du plan), tandis que leurs dessins/calculs automatiques semblent réalisables. Cependant, nous avons décidé de ne pas automatiser la mesure des allèges et des hauteurs des ouvrants, puisque comme nous l'expliquerons en partie I.2.2.3, nous nous concentrerons sur le traitement de représentation 2D des nuages de points, perdant ainsi ce type d'information. Enfin, nous ne travaillerons pas à l'automatisation des processus d'habillage du plan et de mise en présentation, ces tâches sont, en raison de la variabilité de la charte graphique et de leur moindre complexité par rapport à l'extraction des mesures du nuage pour sa vectorisation, moins intéressantes à automatiser.

I.2.2 Chaîne de travail, prise des mesures, traitements et dessins des plans

Les objectifs de l'étude ainsi que les méthodes à disposition étant définis, vient désormais le moment de les confronter aux processus de production des plans du cabinet, afin de connaître les modalités d'applications et développement des traitements à mettre en place.

I.2.2.1 Relevé terrain, caractéristiques des données à traiter

Pour le relevé d'intérieur, le scanner statique Trimble X7 (connecté a une tablette Trimble T10, Figure n°4) est exclusivement utilisé. Il est configuré en mode scan d'intérieur avec prise d'images, une station durant ainsi entre deux et trois minutes, avec ce mode les points mesurés par le scan sont distants les uns des autres, de onze millimètres à dix mètres de celui-ci.

Figure 4 : Trimble X7 et la Tablette Trimble T10 (Source : https://www.buildingpointfrance.fr/nos-solutions/scanners-3d/trimble-x7/)

Le géoréférencement est effectué par l'établissement de stations (quatre au minimum, trois requises et une de contrôle), mesurées avec un récepteur GNSS (méthode NRTK, avec le réseau TERIA) afin de calculer leurs coordonnées dans le système légal (Altimétrie : IGN 69, Planimétrie : RGF93-CC44/CC43, respectivement Alpes-Maritimes et Var). Le nuage de points est ensuite rattaché à ce système via un cheminement (avec le scan) intérieur/extérieur ou extérieur/intérieur, avec mesure de la position des stations par le scan (fonction de mesure de point du Trimble X7).

Une contrainte apparaît : les nuages de points sont géoréférencés dans le système légal français. Ces coordonnées sont ainsi plus "volumineuses" que des coordonnées locales, ce qui, multipliées par les milliers de points constituant un nuage, alourdit considérablement le traitement. Les traitements en Python peuvent être ralentis et, telle que le mentionne Kobena 2023, dans son mémoire, certains logiciels tels que Cloud Compare sont incapables de gérer de si grandes coordonnées et les réduisent donc. Enfin de part, le type de scannage utilisé nous savons que nous devrons utiliser des nuages de points denses, et qu'ils devront par conséquent être sous-échantillonnés afin de limiter les temps de traitements qui notamment, via Python pourrait s'avérer extrêmement long.

I.2.2.2 Logiciels de traitement des nuages de points et de DAO

Pour l'intégralité du traitement des nuages de points (géoréférencement, recalage, affinage, segmentation, orthoprojection, création d'un "publisher"), le cabinet ATRIUM Foncier utilise le logiciel Trimble REALWORKS. L'utilisation de ce logiciel représente un atout majeur. En effet, il intègre un grand nombre d'outils mobilisables qui, grâce à leur intégration dans REALWORKS, peuvent être facilement incorporés aux étapes de production existantes.

Plus spécifiquement, REALWORKS offre de nombreux outils de prétraitement des nuages de points dont nous avons besoin. Il propose d'abord différentes options de filtrage du nuage, permettant de réduire le bruit des nuages de points et de les sous-échantillonner ce qui nous le montrerons en partie II.1.3 s'avère extrêmement utile. De plus, REALWORKS dispose d'algorithmes d'auto-classification et de segmentation qui peuvent s'avérer pertinents. Par exemple, cela nous permettrait de supprimer automatiquement le mobilier, d'isoler les murs, le

sol, le plafond, … (Cf. partie II.1.3). Il est également possible de segmenter manuellement les nuages de points, afin d'isoler, par exemple, les différents étages d'un bâtiment.

Ainsi, REALWORKS est un outil complet qui permet de nous affranchir du recours à des solutions de prétraitement tierces (comme CloudCompare) ou internes (scripts Python, par exemple), limitant ainsi la complexification de la chaîne de production existante, de même que le besoin d'apprentissage de nouvelles compétences par les utilisateurs. En outre, le développement de solutions de traitement de nuages de points 3D en Python s'avérant souvent complexe et peu optimisé, ce qui est problématique lorsqu'il s'agit de traiter des données volumineuses , il a, été décidé que l'intégralité des étapes de traitement des nuages de points 3D se ferait via Trimble REALWORKS.

Quant au DAO le cabinet utilise le logiciel Autocad et le complément Covadis sur trois postes et le logiciel GeogexCAD sur un poste (très similaire a Autocad+Covadis, les fonctions de VRD en moins), empêchant le développement de solution via script des logiciels comme Autocad. C'est deux logiciels de DAO/CAO sont compatibles avec les formats .DWG et .DXF. De plus afin d'exploiter le nuage de points le cabinet importe les orthoprojections (au format .tiff) générer via REALWORKS du nuage dans les logiciels de DAO sous la forme d'image tiff.

I.2.2.3 Parc informatique et ressources de calculs disponibles

Les relevés étant principalement réalisés dans des propriétés privées, l'exploitation des données est délicate et ne peut, pour des raisons de confidentialité/juridique, être transférée sur des serveurs en ligne (par exemple des serveurs tels que Google Collab, offrant gratuitement d'importantes puissances de calcul), ce qui nous contraint à utiliser exclusivement des solutions de traitement local.

Ainsi, nous devons connaître les caractéristiques du matériel informatique de l'entreprise afin de déterminer les ressources de calcul dont nous disposons. Le cabinet dispose de quatre ordinateurs dédiés au traitement de nuages de points et à la DAO. Or, le travail proposé doit de préférence fonctionner sur l'ensemble des postes mentionnés, et doit, donc, être compatible avec la configuration la moins performante, détaillé dans le tableau ci-dessous (Tableau n°1).

Composant	Caractéristiques
CPU	Intel core i7-12700 (2.10 GHz)
GPU	Nvidia T1000 8Go
RAM	32 Go

Tableau 1 : Configuration minimale d'exécution des traitements

Nous le disions en partie I.1, nous avons préféré l'utilisation de méthodes de machine learning. Or, ces dernières nécessitent des calculs rapides, consommateurs de larges ressources graphiques, et plus particulièrement de VRAM. Or, si nous prenons en compte les spécifications graphiques les moins élevées, notre solution doit pouvoir fonctionner avec 8 Go de VRAM, ce qui est relativement peu dans le domaine du machine learning. De plus, même les meilleures configurations du cabinet avec 16 Go de VRAM restent limitées pour l'entraînement de modèles d'apprentissage profond. Cette réalité, combinée au peu de dataset/jeux de données annotées de nuages de points d'intérieur disponibles, restreint nos possibilités de création de modèles, c'est pourquoi nous avons préféré, pour la suite de ce travail, utiliser des modèles pré-entraînés et envisager une solution de type « assistant » (semi-automatique avec option automatique), moins gourmande en ressources. De plus ces ressources limitées nous ont poussé à explorer prioritairement les solutions utilisant des représentations 2D des nuages de points 3D, plus spécifiquement des cartes de densité (cf. Partie II.1). Puisqu'une donnée 3D et outrément plus complexe à traiter, qu'une information 2D.

II Les solutions développées : méthodes, caractéristiques, et fonctionnement

Maintenant que nous avons pris connaissance des outils et méthodes existants, ainsi que des contraintes potentielles découlant du contexte de notre étude, nous présenterons les deux principales solutions développées, l'une utilisant RoomFormer (cf. Partie I.1.1), l'autre construite autour du Segment Anything model (cf. Partie I.1.2).

Cette partie sera, ainsi, organisée de la manière suivante : nous justifierons, d'abord, l'utilisation de projection 2D, nous examinerons ensuite la nature et les résultats de la projection des nuages de points 3D en images 2D. Puis, nous aborderons les prétraitements effectués sur les nuages de points, les outils utilisés afin d'affiner, améliorer les projections Enfin, nous présenterons les deux méthodes de traitement mises en place pour les projections 2D.

II.1 Les pré-traitements, retenus, caractéristiques et fonctionnement

II.1.1 Choix de l'utilisation de projection 2D des nuages de points 3D

Comme nous l'avons vu précédemment, nous disposons de ressources de calcul limitées. Or, une donnée 3D est nettement plus complexe qu'une information 2D. Ainsi, pour simplifier le développement d'algorithmes et répondre aux contraintes en ressources (I.2.2.3), nous ne vectoriserons que des projections/coupes de ces nuages sur des plans 2D (vue du dessus). Cette décision a été inspirée par le process développé dans le TFE « Mise en place d'un processus de dessin automatisé de plans d'intérieurs à partir de nuages de points acquis par LIDAR », de Talec 2017. En effet, comme l'explique et l'illustre, le dit mémoire, la perte d'information due au passage de la 3D à la 2D est finalement limitée, tandis que la perte de complexité des données est significative. De plus, le dessin manuel de plans d'intérieur s'effectue déjà par l'intermédiaire de coupes 2D dans le nuage de points 3D.

II.1.2 Nature, fonctionnement et caractéristique de la projection 2D des nuages de points 3D

II.1.2.1 Choix de la carte de densité

Mais si le TFE de Talec 2017 utilise des ortho-projections/orthoimages nous avons décidé d'utiliser, comme représentations 2D des nuages de points 3D, des cartes de densité. En effet, l'ensemble des méthodes de reconstruction de plan d'intérieur (MonteFloor, Stekovic 2021, HEAT, Chen 2021) dont RoomFormer (I.1.2) est l'aboutissement dernier, utilise de telles représentations 2D. Représentations qui nous ont semblé, pour des raisons que nous allons développer, plus adaptées et efficaces que les orthoimages.

Pour commencer, une carte de densité consiste en un affichage de la distribution des points dans un espace bidimensionnel, indiquant par l'intensité des couleurs les zones de plus forte densité (de points). La projection étant réalisée selon un plan horizontal (parallèle au sol), on comprend que les murs, qui se matérialisent dans un nuage par un amas de points verticalement répartis, ressortent particulièrement. Or, il est essentiel, afin d'obtenir une vectorisation automatique efficace d'un nuage de points, d'identifier convenablement la position des murs. En effet, l'estimation de la position de ces derniers déterminera la qualité de la segmentation des pièces et donc la précision du calcul des superficies et, plus globalement, du plan général. Ainsi, nous avons préféré les cartes de densité aux orthoimages, puisque les orthoprojections ne font pas ressortir aussi clairement les murs que les cartes de densité. Dans une carte de densité, les murs se distinguent par leur couleur plus intense, tandis que dans une ortho-image, les couleurs sont plus uniformes ou non visibles si les plafonds subsistent, cf. Figure n°5, ci-dessous.

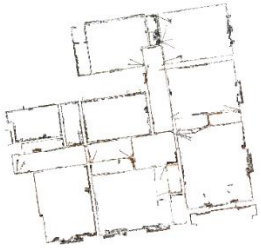
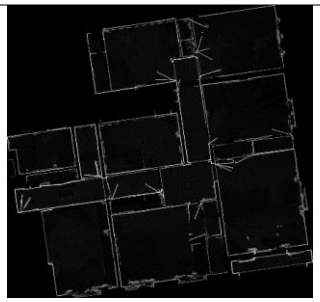
Orthoimage (avec murs, plafonds, sols)	**Orthoimage (avec murs)**	**Carte de densité (1080p*1080p)**
		

Figure 5 : Les différents types de projection 2D envisagées

De plus une orthoprojection présente plus de détails et d'éléments qu'une carte de densité (meilleure visibilité du mobilier, des ouvrants), des détails qui s'avèrent superflus, alourdissant l'exécution des algorithmes, voire dégradant leurs résultats (détection, dessin, d'éléments non désirés).

Autre point, lors de l'orthoprojection via REALWORKS la résolution de l'image produite est variable. Puisqu'il est nécessaire de définir le PPI (Pixels Par Pouce), l'usage dans le cabinet étant d'environ 3.5. Avec un PPI trop bas, on obtient des images peu définies et finalement moins exploitables que des cartes de densité. Un PPI élevé, quant à lui, génère des orthoprojections de grande résolution, or les modèles de machine learning consomment des ressources de calcul exponentiellement à la résolution des images d'entrées.

Enfin, la carte de densité permet plus de post traitements des images. Post traitements qui s'avèreront très utile pour améliorer la précision des résultats, Partie III.2. Ainsi, la carte de densité a été préférée comme support de projection 2D du nuage de points 3D. Elle permet une représentation simple des éléments essentiels et reste alors efficace pour des rapports superficie représentée/définition plus grands.

II.1.2.2 Méthode de création carte de densité

Pour créer des cartes de densité, nous utilisons une version modifiée du script de projection présent dans le répertoire GitHub du projet RoomFormer, précédemment montré. Les fonctions generate_density, export_density proviennent de ce répertoire et sont visibles en annexe 16, p51. Ces éléments étaient originellement écrits pour générer les cartes de densité à partir des nuages de points du dataset Structure3D, des modifications ont été apportées afin de permettre l'utilisation de simple nuage de points aux formats. LAS, exporter depuis le logiciel Trimble REALWORKS. Le fonctionnement général et décrit dans le diagramme, Figure n°6, ci-dessous.

Ce script prend, donc, en entrée un nuage de points, une résolution (celle de la carte de densité générée) et un chemin de fichier de sortie. Il est à noter le script prend en entrée des fichiers de nuages de points Las, pour les convertir immédiatement en. Ply. Ce choix est dû à notre volonté de traiter les nuages de points aux formats. Ply (formats répandus, notamment pour les dataset, et facilement manipulable en python) alors que REALWORKS ne permet pas l'export des nuages de points dans ce format.

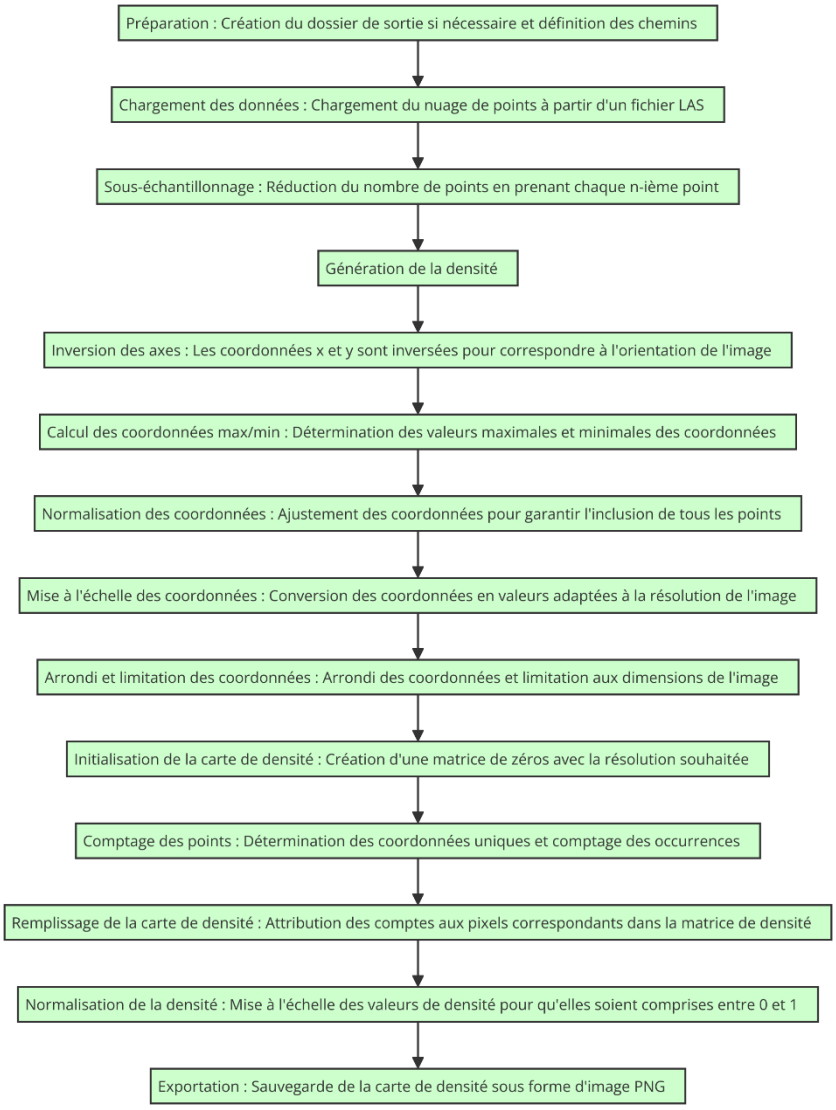


Figure 6 : Diagramme détaillant le fonctionnement de la projection du nuage de points 3D en carte de densité 2D

L'image produite n'est directement géoréférencée, les prédictions ultérieurement générées le seront donc dans l'espace image. Ainsi pour pouvoir géorefenrencées les prédictions, les coordonnées de chaque pixel dans le repère du nuage sont estimées lors de la projection et sauvegarder dans un fichier. npy (fichier numpy, format choisit car facilement manipulable en python), cf.Annexe 9, 10, 11 et 16, p45, 46, 51. Les prédictions des modèles sont en effet exprimées dans le repère image et peuvent grâce aux fichiers. npy être replacées dans le repère du nuage de points 3D

Deux points sont à noter : premièrement, nous n'avons pas ajouté les coordonnées géoréférencées de chaque pixel dans le fichier image, afin d'éviter d'alourdir les traitements, d'autant plus que les coordonnées RGF93 sont volumineuses (cf. Partie I.2.2.1).

Deuxièmement, nous avons opté pour une résolution fixe (comme brièvement abordée en fin de Partie II.1.2.1) afin de nous conformer aux exigences des modèles de machine learning. En effet, de nombreux modèles (dont RoomFormer) ne fonctionnent correctement qu'avec des images de même résolution que celles utilisées pour leur entraînement. Et, dans les cas où la résolution est libre (le Segment anything model, par exemple, accepte toutes les résolutions), les ressources nécessaires à l'exécution augmentent exponentiellement avec la taille des images. Dans notre contexte, cela nous oblige à limiter la résolution des cartes de densité traitées, nous interdisant donc des résolutions dynamiquement définies (risque d'image trop grande).

Enfin l'utilisation de projection 2D des nuages de points nous obligent à isoler les étages des bâtiments concernés. En effet, la projection se faisant sur un plan horizontal au-dessus du nuage, il ne faut, pour que la carte de densité soit exploitable/compréhensible, qu'il n'y est que la structure d'un unique étage projeté. Nous n'avons pas cherché à automatiser cette étape. Cependant, le cabinet travaillant déjà avec des projections 2D des nuages (ortho image), il est déjà d'usage d'isoler les étages des bâtiments.

Pour plus de détail sur le fonctionnement du script de projections il est visible en Annexe 16, p51.

II.1.3 Pré-traitement des nuages

II.1.3.1 Sous-échantillonnage

Nous avons montré dans la partie I.2.2.1, que les nuages traités (car issus d'un scanner terrestre LIDAR, Trimble X7) sont denses. Cependant, pour la création d'une carte de densité, il n'est pas nécessaire de projeter autant de points. En effet, il est possible d'obtenir la même quantité d'informations en 2D avec moins de points. Sachant qu'un volume de données aussi important ralentit considérablement le processus de création de la carte de densité 2D (cf. partie II.1.1), nous procédons donc à un sous-échantillonnage du nuage via la commande Echantillonnage de REALWORKS.

Nous avons déterminé empiriquement qu'un pas d'échantillonnage de plus 0,05 cm dégradait trop fortement la précision. Au-dessus, la perte de données est conséquente (cf. Figure n°7).

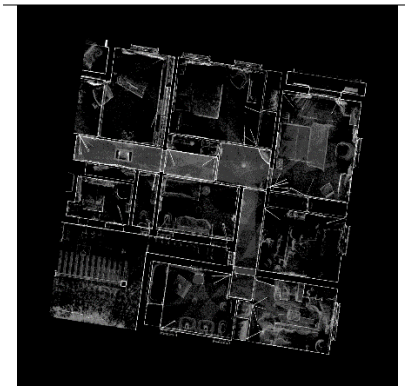
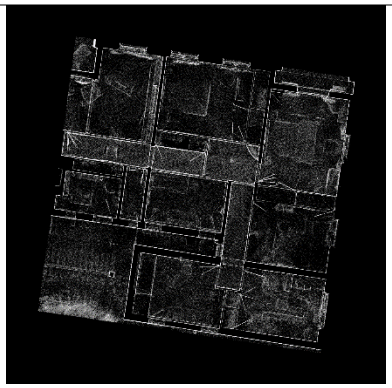
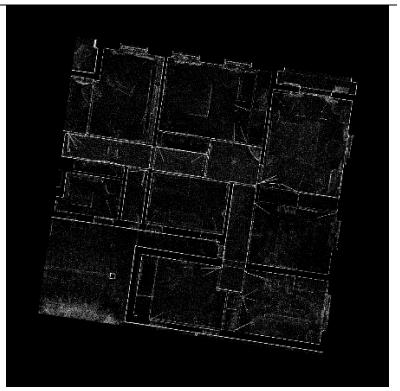
Carte densité (échantillonnage 1 cm)	Carte densité (échantillonnage 5 cm)	Carte densité (échantillonnage 7 cm)
		

Figure 7 : Carte de densité issue de nuage de points 3D échantillonnés selon diffèrent pas d'échantillonnage

A noter que l'ensemble des cartes de densités présentées (sauf indication contraire), on suivit les post traitements décrit en partie III.2 et proviennent d'un nuage de points échantillonnés avec un pas de cinq centimètres, de même pour les cartes de densités utilisées en inférence.

II.1.3.3 Classification et segmentation du nuage de points 3D

Bien que l'algorithme RoomFormer ne nécessite pas de segmentation ou de classification préalable du nuage, et voit même ses résultats dégradés par un tel nettoyage (cf. partie III.1), l'outil développé avec le modèle SAM, lui, bien qu'il fonctionne sans cela, à des performances considérablement meilleures (notamment en automatique,) après la classification des murs, sols et plafonds. En effet, une telle classification nous permet de supprimer le mobilier et donc d'améliorer la précision de la détection des murs. Cependant les améliorations sont plus modestes pour le fonctionnement semi-automatique (invite sous la forme de clic)

Le choix de l'outil de segmentation a été assez simple, REALWORKS possédant justement un outil de classification des nuages de points d'intérieurs, entraîné sur les classes murs (Wall), sol (Floor), plafond (Ceiling), autre (Other). En effet, ce modèle nous a offert des résultats tout à fait satisfaisants, et répondait, par son intégration à REALWORKS, à tous les

critères exposés en partie I.2.2.2 (le cabinet est déjà familier de l'outil, logiciel commercial suivi et soutenu par ses développeurs, solution performante et ergonomique…).

II.2 Solution et méthodologie de traitement des cartes de densités

II.2.1 Reconstruction de plan d'intérieur via RoomFormer

Initialement, l'objectif était d'automatiser au maximum (avec peu ou pas d'intervention humaine), le processus de vectorisation du nuage de points 3D en plan 2D. Nous recherchions donc des solutions de vectorisation 2D de nuages de points 3D, utilisant des supports 2D, orientés, de préférence, vers la production de plans d'intérieur. Or, comme nous l'avons vu dans la partie I.1.4, la méthode RoomFormer répond parfaitement à ces critères : elle utilise des projections 2D (cartes de densités) et est orientée vers la génération de plans d'intérieur, en se focalisant sur la détection et la segmentation des pièces. Le fonctionnement général de cette méthode correspond à celle décrite à la Figure n°3, page 11.

II.2.1.1 RoomFormer, implémentation Python

RoomFormer dispose d'une implémentation Python officielle, disponible sur GitHub. Nous avons utilisé cette dernière afin d'exploiter la méthode présentée dans l'article du même nom (cf. I.1.2). En effet, étant peut familier de la programmation pytorch et plus généralement de la programmation de modèle de machine learning en Python, il nous semblait irraisonnable de créer le modèle à partir de zéro, alors même que nous ignorions s'il s'avèrerait efficace sur les données que nous comptions traiter.

Cependant, cette intégration Python étant conçue uniquement pour traiter les données des datasets Structured3D et SceneCAD nous avons, quand même dû écrire une version modifiée du script d'inférence. La modification de l'intégration du modèle en PyTorch fut plus longue que prévue, étant, à ce moment peu connaisseur du fonctionnement de PyTorch.

De plus, l'utilisation de ce répertoire GitHub en local nous obligea à installer les bibliothèques Python requises, ce qui, comme nous allons le voir, a posé de nombreuses difficultés.

II.2.1.2 Mise en place d'un environnement d'exécution

Le répertoire GitHub fournit des fichiers exécutant l'installation des bibliothèques et répertoires nécessaires à son bon fonctionnement. Seulement, l'un de ces fichiers est au format script shell, format Linux.

En effet, cette intégration Python a été conçue pour fonctionner sur un environnement Linux et n'a pas été testée sur Windows. Nous avons alors, dans un premier temps, tenté d'installer les éléments manuellement, sans succès, et avons, finalement, décidé d'utiliser un WSL (Windows Subsystem for Linux).

Nous avons opté pour une distribution Ubuntu (étant la plus répandue, suivie et soutenue), sur laquelle nous avons installé Anaconda afin de créer un environnement adapté. Enfin, dans cet environnement, nous avons installé les versions de Cuda11, torch 1.9.0 et torchvision 0.10.

Cette installation fut complexe et laborieuse (même une fois le système Linux, Ubuntu WSL, mis en place, des problèmes de compatibilité entre les packages et de support de CUDA par la carte graphique subsistèrent), et ce, sur deux postes différents. De plus, l'utilisation d'un WSL Linux oblige l'utilisation d'Anaconda via ligne de commande, ce qui réduit grandement l'ergonomie et l'accessibilité de la chose. Ainsi, il apparaît déjà que cette implémentation de RoomFormer, bien que prometteuse, est difficile d'accès et d'utilisation, alors que nous souhaitions justement une méthode simple et ergonomique.

II.2.2 Segmentation des cartes de densités via l'utilisation de SAM

Nous le verrons lors de la présentation des résultats du modèle RoomFormer en partie III.1, il s'est avéré peu performant. Nous avons alors d'abord envisagé des modèles de segmentation d'image afin de prétraiter les cartes de densité. Cependant, la méthode développée s'est vite révélée suffisamment performante pour faire office d'outil de vectorisation des cartes de densité à part entière. En effet face aux résultats décevants et aux limitations non envisagées (au départ) de RoomFormer (que nous verrons en partie III.1), nous nous sommes rabattus sur des méthodes de segmentation/classification d'image plus générale.

Mais, comme nous l'avons expliqué nous cherchions des modèles disposant de version pré-entrainée. Or la nature des images (carte de densité) que nous traitons et assez particulière, de sorte qu'aucun modèle, autres que ceux mentionnés en I.1.2 (du moins nous n'en avons pas trouvé) n'a été entraîné ou conçue, sur/pour ce type de données (ou sur un type proche). Nous avons pour ces raisons, plutôt chercher, des modèles de machine Learning de segmentation d'image capables d'une bonne généralisation (zéro-shot performance). Et, comme expliqué en partie. I.1.1, le Segment Anything Model (cf. I.1.1), publiait de façon open-source par l'entreprise Meta, nous est apparu être le candidat idéal. Il est doté d'excellente capacité de généralisation, et est tellement performant qu'il est désormais inclus dans de nombreux outils d'annotations d'images. Autre atout majeur, Meta, a fourni une API Python avec son modèle, rendant sont utilisation très aisée.

II.2.2.1 Forme de l'outil à créer

En partie. I.1.3 nous indiquions que le modèle SAM, accepte deux types d'entrées : pointage, boîte englobante, plus un mode de segmentation entièrement automatique. Ce mode fonctionne de la manière suivante, SAM crée automatiquement une grille de point, et exploite ensuite chaque point comme il utiliserait des entrées par pointages manuels.

L'idée fut donc de créer un outil d'assistance à la DAO/CAO depuis les cartes de densité des nuages de points, avec interface graphique, fonctionnant par pointage de l'intérieur des pièces via des cliques et si les ressources de la machine le permettent une possibilité de segmentation automatisée des cartes de densités. L'outil doit aussi permettre la visualisation en direct des vecteurs/polylignes créer, avec possibilité de modifier la position et/ou de supprimer les dites polylignes. Le process développé est décrit via le diagramme ci-contre, Figure n°8

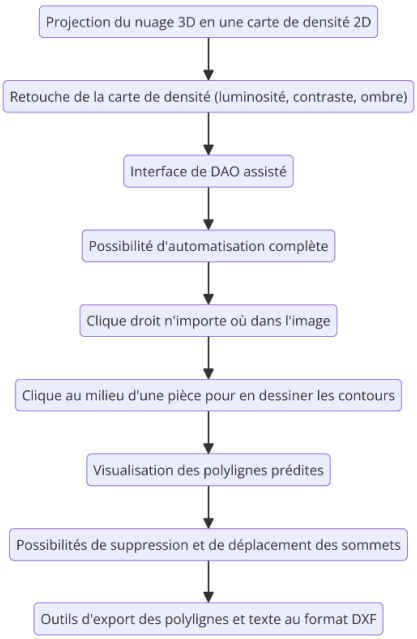


Figure 8 : Diagramme de l'assistant à la DAO via SAM

II.2.2.2 Bibliothèques utilisées et création de l'environnement Python

Pour la partie graphique de l'assistant DAO, nous avons choisi d'utiliser la bibliothèque Python PyQt5. En effet, celle-ci, est plus complète et fournie que la bibliothèque graphique Python de référence, Tkinter, tout en étant aussi documenté. Ainsi, elle permet, comme Tkinter de bénéficier d'une large documentation tout en créant des interfaces plus esthétiques, complètes et ergonomiques. En effet, le fonctionnement axé widget de PyQt5, est particulièrement adapté à la forme de l'interface voulue (multiples fenêtres).

Puisque, l'interface se composera d'une fenêtre supérieure affichant l'image et les polygones prédits, avec la possibilité de les supprimer et/ou de déplacer leurs sommets. Une fenêtre inférieure affichera la liste des cartes de densité chargées, avec la possibilité de sélectionner celle à afficher. Les deux fenêtres seront réglables en hauteur l'une par rapport à l'autre. Enfin, des boutons seront disposés en bas de l'interface, permettant la suppression de polylignes, l'export en DXF des polylignes, et le chargement des cartes de densité.

Nous avons d'abord construit cet outil autour de l'API officielle de SAM. Ainsi, nous devions nous assurer au préalable de disposer, sur les postes testés, des versions compatibles de Torch, Torchvision et CUDA. Cependant, nous avons rencontré à ce moment-là des problèmes (installation des modules Torch et CUDA laborieuse) similaires à ceux rencontrés avec RoomFormer (décrit en partie II.2.1.1). De plus, avec cette API, le mode automatique de SAM était assez gourmand en VRAM, nécessitant 16 Go pour une carte de densité de 1080p*1080p, et l'inférence par pointage (clic) était assez peu agréable d'utilisation, avec un délai de trente secondes à une minute entre le clic et l'apparition de la polyligne détectée.

Cependant, nous avons trouvé une bibliothèque intégrant SAM, plus simple à installer et à utiliser, et offrant de meilleures performances : Ultralytics. En effet, Ultralytics est une entreprise américaine qui met à disposition du public (sous la forme d'une bibliothèque Python) des modèles de détection, de segmentation et de classification d'image, YOLOV. Ainsi, dans le cadre du développement de ces modèles, Ultralytics a intégré le modèle SAM (notamment pour faciliter la création de dataset), à sa bibliothèque, et ce dans quatre versions : légère, moyenne, large et mobile (destinées à l'usage des appareils mobiles). Nous discuterons de la version du modèle retenue en partie III.3.1. L'utilisation de cette bibliothèque, en lieu et place des versions officielles mentionnées au paragraphe ci-dessus, présente de nombreux avantages.

Si elle nécessite l'installation de Torch, Torchvision et CUDA, elle est compatible avec une large gamme de versions. Enfin, le mode automatique de SAM est beaucoup moins exigeant en ressources graphiques (tous les temps d'inférence sont globalement réduits). En effet, sur une carte de densité de 1080p*1080p (la même qu'évoquée dans le paragraphe précédent), il fonctionne sur un PC doté de seulement 6 Go de VRAM, contre 16 Go (minimum requis) précédemment.

II.2.3 Post-traitement des masques prédits par SAM

Maintenant que nous avons présenté le fonctionnement de l'interface et ses soubassements techniques (le modèle de machine learning, bibliothèques Python …), nous allons expliquer la nature et la raison des post traitements appliqués au masque de sortie du modèle SAM. En effet, le modèle renvoie des masques aux contours irréguliers et complexes, visibles sur la figure n°11 p.29.

Ainsi, nous présenteront les trois post-traitements que nous avons implémentés dans leurs ordres d'exécution. Dont voici un résumé dans le diagramme ci-contre, Figure n°9, ci-contre.

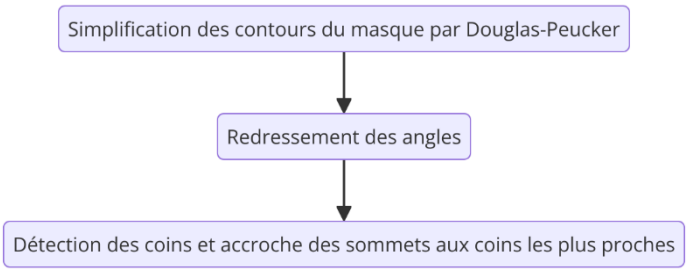


Figure 9 : Diagramme du post-traitement des masques SAM

Il est à noter que la biothèque ultralytics renvoie les masques prédits sous forme de polygones, c'est à dire de matrice (np. Array) ou chaque ligne correspond aux coordonnées d'un sommet dans l'espace image (coordonnées en pixel).

II.2.3.1 Simplification des contours du masque

Nous avons vu qu'il était nécessaire de simplifier le contour des masques prédits. Pour effectuer cette tâche l'algorithme de Douglas-Pecker nous a semblé être une solution adaptée. En effet, cet algorithme fonctionne en supprimant les points qui sont à une distance inférieure à un seuil spécifié par rapport à une ligne droite formée par deux points extrêmes. L'algorithme est récursif et continue à diviser la courbe jusqu'à ce que tous les points restants soient à une distance supérieure au seuil de la ligne droite formée par leurs points extrêmes.

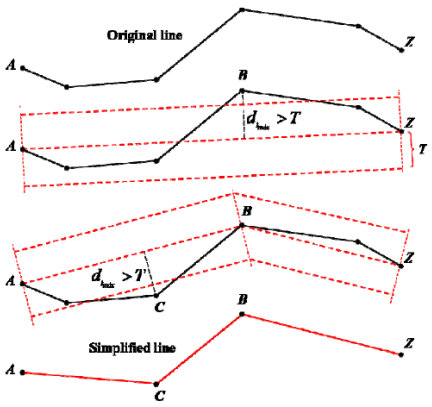


Figure 10 : Principe de la méthode de Douglas-Peucker (Source : https://www.researchgate.net/figure/Example-of-the-application-of-de-Douglas-

Concrètement l'intégration Python de cet algorithme c'est fait via l'utilisation de la fonction approximate_polygon de skimage (qui utilise justement cet algorithme). Dans le code la distance seuil est définie par une variable nommée tolérance, elle définit la force de la simplification (pointillé rouge dans la figure n°10).

Comme le montre la Figure n°11, ci-dessous, l'application de cette fonction simplifie le masque, et ce proportionnellement à la valeur de la distance seuil (tolérance dans le code). Les cercles rouges représentent les sommets des polygones verts.

Douglas-Peucker, tolérance 0	**Douglas-Peucker, tolérance 2**	**Douglas-Peucker, tolérance 5**

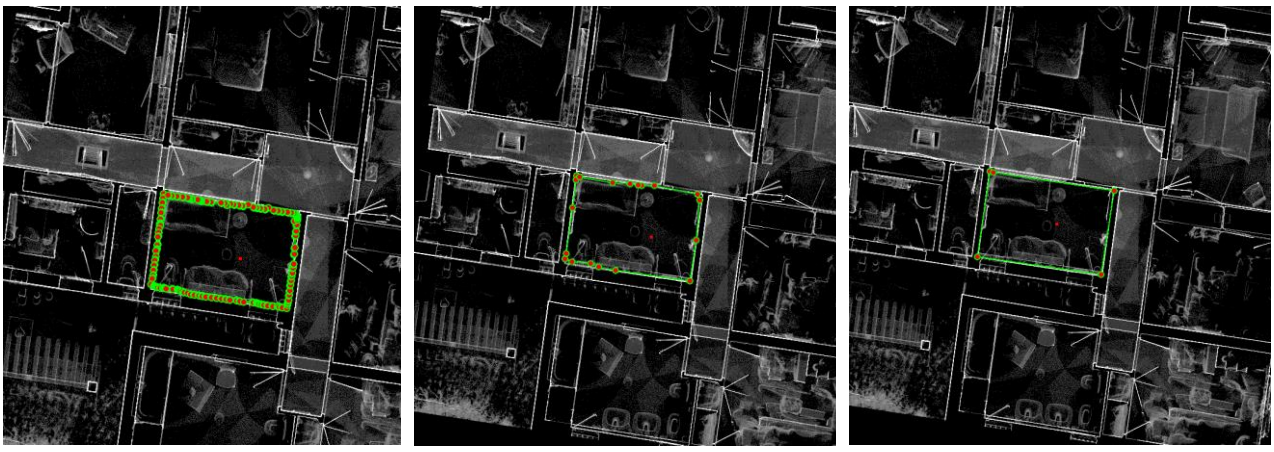


Figure 11 : Impacte de la tolérance sur la simplification de Douglas-Peucker

II.2.3.2 Redressement/ajustement des angles

Nous ajustons ensuite les angles des polylignes. En effet, nous pouvons supposer que dans le contexte du dessin d'intérieur la plupart des pièces possèdent majoritairement des angles proches de l'ange droit.

Ainsi pour ajuster les angles d'un polygone et les rapprocher des angles droits, nous devons déplacer légèrement les sommets du polygone. L'idée est alors de calculer l'angle formé par chaque triplet de points consécutifs du polygone, et s'il est proche d'un angle à 90° (seuil paramétrable), de déplacer le point central du triplet, de sorte à réduire l'écart entre l'angle calculé et un angle droit. Mais, ce déplacement doit être minimal pour ne pas déformer significativement le polygone d'origine.

Nous suivons alors les étapes suivantes, illustrées en Figure n°12, ci-dessous :

1. **Calcul de l'angle** : Par exemple, pour chaque triplet de points (A, B, C), dans le polygone, nous calculons l'angle formé en B. Cet angle est calculé en utilisant les vecteurs AB et BC.

2. **Ajustement du point central** : Si l'angle est proche de 90 degrés (dans une certaine tolérance, variable à définir manuellement), nous déplaçons le point B pour rapprocher l'angle de 90 degrés. Le déplacement se fait de manière à minimiser le changement global dans la forme du polygone. Concrètement un vecteur de déplacement est calculé pour déplacer le point central de l'angle de manière à le rapprocher de 90 degrés. Le vecteur de déplacement est une combinaison des vecteurs perpendiculaires aux les deux vecteurs formant l'angle.

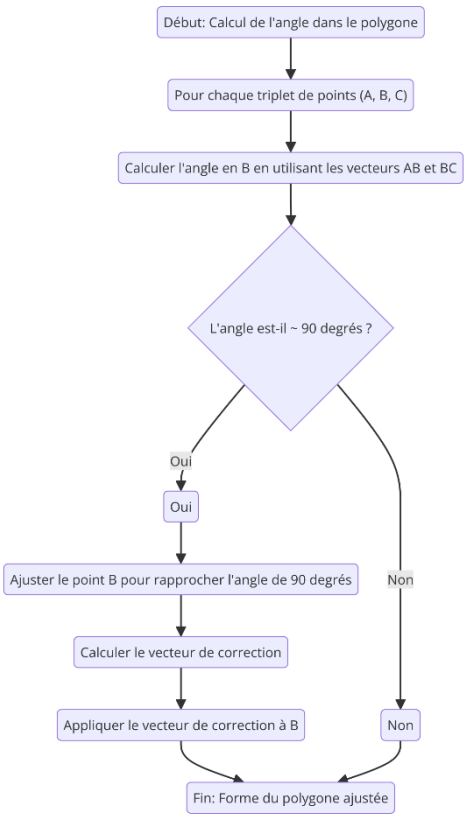


Figure 12 : Diagramme de l'ajustement des angles des masques issues du modèle SAM

3. **Direction de déplacement** : Application du vecteur de correction au point B.

II.2.3.3 Détection des coins et accroche des sommets aux coins les plus proches

Enfin, une fois les contours du masque simplifiés et les angles ajustés, nous obtenons globalement de meilleurs résultats, cependant à cette étape, certains sommets des polygones sont proches des coins de la pièce, sans y correspondre.

Pour remédier à ce problème nous avons ajouté une étape de détection des coins de l'image (carte de densité du nuage de points), pour ensuite attacher, les sommets des polylignes les plus proches à ces coins, si et seulement si la distance est inférieure à un seuil en pixel, à définir manuellement.

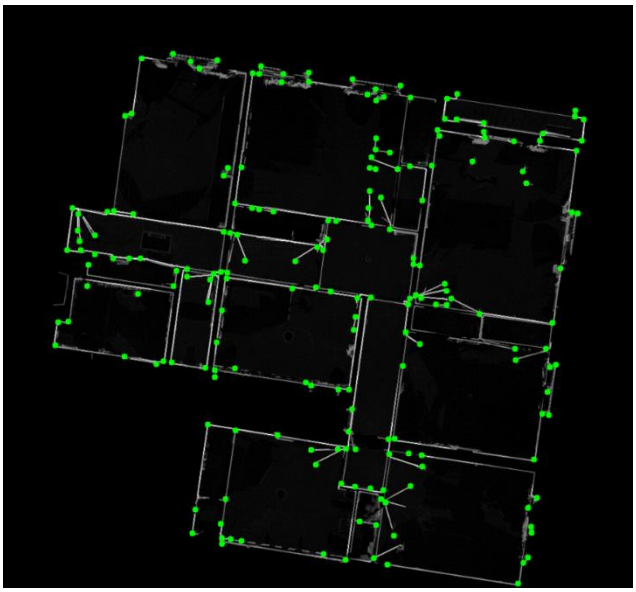


Figure 13 : Détection des coins Shi-Tomasi sur une carte de densité (carte de densité post traité selon la méthode décrite en partie III.2.2)

Nous utilisons la détection de coins de Shi-Tomasi (cf. Figure n°13, ci-dessous), préférer à la détection de coin de Harris, car moins sensible au bruit, et pour ces paramètres d'entrés plus adaptés à nos besoins. Puisque pour ce type de détection de coins trois paramètres doivent être entrées, le nombre de coins maximums, la distance minimale entre les points détectés, et le niveau de qualité minimum accepté. Nous avons empiriquement déterminé des paramètres par défaut (cf. III.2.2), tous les traitements présentés dans ce mémoire le seront avec le paramétrage par défaut.

III Analyse et comparaison des résultats

III.1 Qualité des prédictions du modèle RoomFormer

Nous allons désormais observer la qualité des prédictions du modèle RoomFormer. RoomFormer présenta, en mode Richfloorplan (détection des ouvrants et du type de pièce), des résultats peu satisfaisants avec les données datasets, et convenables en mode standard.

Tandis que sur nos nuages de points les résultats sont incomplets et très peu précis. Peu de pièces sont détectées et leurs contours sont très imprécisément tracés. Cependant, sur ses données le mode Richfloorplan offre de meilleurs résultats. L'article Floorplan Generation from Noisy Point Cloud, Talotta 2024, nous a offert une piste d'amélioration. Puisque cet article explique que le modèle RoomFormer est extrêmement sensible au bruit, car il a uniquement été entraîné sur des nuages de points synthétiques (dataset structure 3D et SceneCAD), donc par nature dénué de bruit. Ainsi, les chercheurs ont affiné le modèle (fine-tuned), avec un dataset (FloorNet), constitué de nuages issus de mesures terrains via scanner LIDAR terrestre. Le fichier de poids résultant est disponible en ligne (via le répertoire GitHub). Il existe deux versions de ce fichier, une pour le mode standard, l'autre pour le mode Richfloorplan. Nous avons constaté que le fichier poids, Richfloorplan offrait à nouveau les meilleurs résultats, sur nos données. Ces résultats, sont notamment, significativement meilleurs que ceux, obtenus via les fichiers de poids fournit avec le modèle. Cependant, comme nous pouvons le voir dans le Figure n°14, ci-dessous, les résultats subsistent peu précis et incomplets.

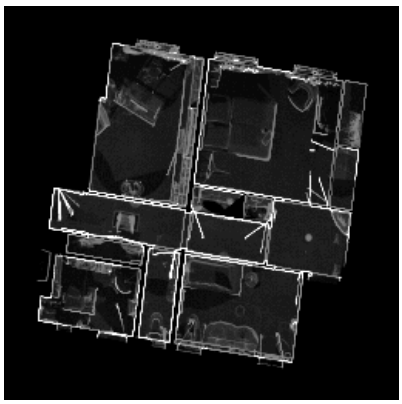
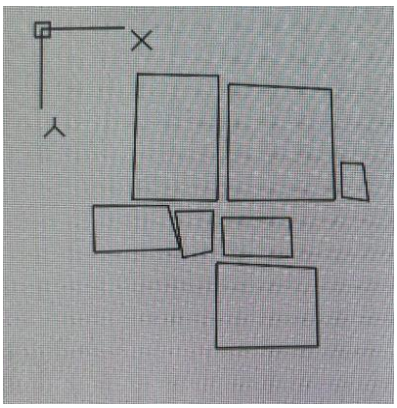
Carte de densitée fournit à RoomFormer	**Piéces predites par RoomFormer**
	

Figure 14 : Résultat de RoomFormer sur notre nuage test

De plus nous ne nous pouvons avec ces fichiers de poids, n'inféré que des cartes de densités de dimension 244p*244p (dimension des images d'entraînement). Ce qui limite grandement la superficie des nuages traitables. En effet, si le nuage est trop grand la carte de densité dû à sa faible résolution, sera peu lisible et compréhensible (écart entre les murs non visibles, petite pièce n'apparaissant quasiment pas ...). Afin de remédier à ce problème nous devrions entraîner le modèle sur des images de plus fortes définitions, ce qui est extrêmement gourmand en ressource (24Go de VRAM selon Yue 2023, sur des images de seulement 244p*244p), tandis qu'au vu des résultats existants sur des cartes de densité de 244p*244p, nous n'étions pas certains d'obtenir des vectorisations plus précises sur des images de plus haute définition. Ainsi, si, nous avons dans un premier temps tenté de trouver un moyen de segmenté automatiquement, en fonction des pièces, une image (carte de densité) de haute résolution (plus de 244p*244p), en plusieurs petites vignettes de 244p*244p, nous avons, cessé d'essayer d'améliorer les résultats de cette méthode. En effet, au début envisagé comme pré-traitement à RoomFormer, SAM (solution de vignetage envisagée), aux vues de ces bonnes performances en segmentation, a finalement complètement remplacé RoomFormer dans le process d'automatisation du traitement, avec de bien meilleurs résultats.

III.2 Étude et comparaison de la qualité des résultats et prédictions des modèles et méthodes en fonction des différents traitements des cartes de densité

Avant d'analyser les résultats du segment anything model, nous nous attarderons sur les manipulations des cartes de densité susceptibles d'améliorer les performances du model. En effet, les cartes de densité que nous générons (Partie II.1.2.2 pour plus de détails), sont de simples images, et peuvent donc faire l'objet de divers traitements. Notre objectif étant de vectorisés les éléments tels que les murs, les ouvrants, les poutres, il serait intéressant de traiter les images/carte de densité afin de faire ressortir ces caractéristiques. Nous avons alors constaté que l'augmentation de la luminosité, de l'exposition, et de la netteté, permet de faire ressortir les murs et ouvrants.

III.2.1 Impacte sur la prédiction des masques SAM

Cependant, il est apparu après de multiples tests, qu'une trop grande mise en avant des contours des pièces/murs (seuillage de l'image, augmentation des contrastes, carte de densité issu uniquement du nuage de points des murs …), aux détriments des autres éléments de l'image, dégrade les performances du modèle, comme on peut le voir sur le Figure n°15.

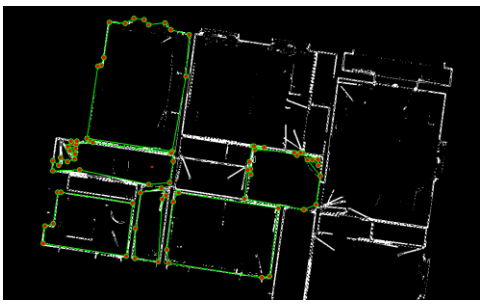
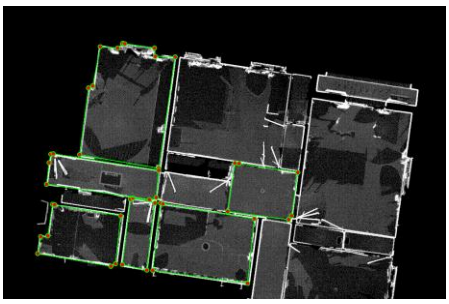
SAM sur carte de densité seuillée	SAM sur carte de densité traitée selon III.2
	

Figure 15 : SAM sur différents types de cartes de densité

En effet, ceci s'explique par le fonctionnement du modèle et la nature des données d'entraînement. Le segment anything model est, comme expliqué en partie I.1.1, un modèle de segmentation d'image entraîné sur un énorme jeux de données, principalement photographique. Il est donc conçu et entraîné pour segmenter des objets/formes plein/pleines (exemple : détecter les contours d'un chien dans une image). Or, des images comme celles des cartes de densité seuillées (Figure n°15), présentent un ensemble noir formé de traits ponctuels, apparence peu commode pour SAM.

Ainsi nous devions afin d'améliorer les performances de SAM faire ressortir les murs tout en opacifiant l'intérieur des pièces. Ceci nous oblige donc à créer les cartes de densité avec des nuages de points au moins formés des murs (délimitation des pièces), du sol et du plafond (servant à opacifier l'intérieur des pièces). En effet, si sur la carte de densité que nous obtenons en sortie de script, les points des sols et plafond (du fait de leurs plus faibles densités) , sont peu visibles, il est possible de les faire ressortir en traitant l'image. Nous avons, en effet observé que la diminution du contraste des cartes de densité, l'augmentation de leurs expositions et l'accentuation de leurs ombres, ainsi que l'augmentation de leurs luminosités permettent de bien faire ressortir les points des sols et plafonds des nuages de points 3D sources, tout en gardant des murs bien visibles (Figure n°16).

Carte de densité originale (nuage non classifié)	Carte de densité retouchée (nuage non classifié)

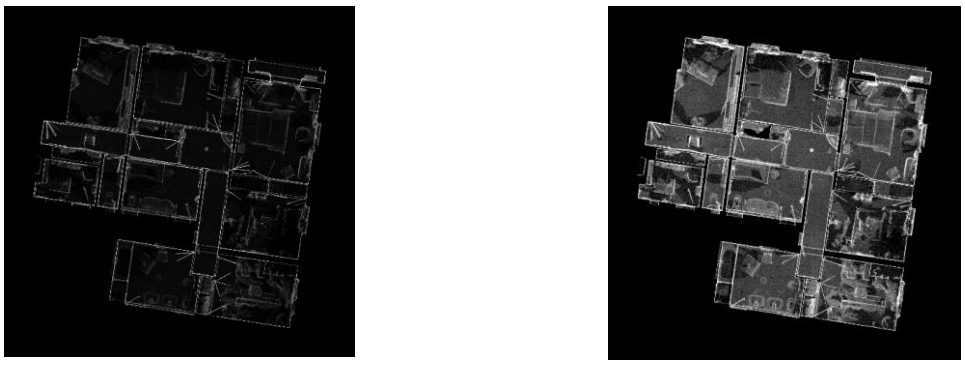


Figure 16 : Carte densité originale et retouchée

Il est à noter, qu'en cherchant à mettre en avant les points du nuage autre que les murs, le mobilier est bien plus apparent (cf. Figure n°14, ci-dessus). L'utilisation de carte de densité retouchée suivant les prescriptions précédemment édictées, encourage donc à la classification/segmentation préalable du nuage afin d'isoler les éléments mobiliers.

Enfin, l'ensemble des retouches ont été effectuées via l'application Photos Windows. Nous avons tenté d'intégrer directement ces retouches à notre script de génération des cartes de densité. Cependant, les résultats étaient beaucoup moins satisfaisants que ceux obtenus via l'application Windows Photo. Nous avons donc continué d'utiliser le logiciel Windows Photos.

III.2.2 Amélioration de la détection des coins

Comme expliqué en partie II.2.3, nous appliquons des post traitements aux masques prédits par le modèle SAM. Or, la détection des coins des pièces est dégradée par la présence de mobiliers dans le nuage, mobilier mise en avant par les post traitement en partie II.2.3. Nous préférons donc utiliser des cartes de densité issue de nuages classifiés (nettoyés du mobilier). Quant aux paramètres de shi-Tomassi, les valeurs idéales varient selon la nature des nuages. Cependant, nous avons constaté que les valeurs suivantes offraient des résultats robustes et polyvalents, maximum de coins à 1000, qualité des coins seuilles à 0.15, distance minimale entre les coins à 10, les résultats de la figure n°14, p.32, suivent ce paramétrage.

III.3 Analyse de la qualité des prédictions du modèle segment anything model, SAM

Notons que nous utilisons la version prée-entraînée la plus large du segment anything model, fourni par Ultralytics. En effet, ce dernier offre une précision significativement meilleure, tandis que l'augmentation du temps d'inférence et de la puissance nécessaire est relativement raisonnable.

Ainsi, afin d'évaluer la qualité des masques obtenus, nous allons comparer la forme des pièces et leurs superficies obtenues après exécution, sans modification. Nous examinerons ici des résultats obtenus via invite manuelle, sans nous attarder spécifiquement sur le mode automatique. En effet, ici nous discutons de la qualité des masques, polyligne produites, or la précision de ces dernières varie peu entre les différentes modalités d'invite, la vraie différence étant que la segmentation automatique a tendance à « sur-segmenter » les cartes de densités, segmentation du mobilier, d'espace vide entre les pièces …

Commençons par observer la forme des éléments. Nous pouvons voir sur la Figure n°15, p34, que bien que relativement juste les polylignes semblent souvent être légèrement décalées vers l'intérieur des pièces. Cela est dû aux masques bruts fournis en sortie du segment anything model, en effet les masques de SAM, définissent/détectent mal les angles aigus. Or, les coins d'une pièce sont souvent proches d'un angle droit, les coins sont alors mal rendus par les masques, qui arrondir les angles. Dès lors, lors de l'étape de simplification, si la tolérance et faible, l'angle va être dessiné en biseau, et la polyligne alors, bien que plus précise sera plus irrégulière. A l'inverse si la tolérance de simplification est élevée, les polylignes seront plus rectilignes, mais ce sera alors l'un des deux points dessinant le biseau qui sera défini comme le sommet l'angle, décalant alors une partie de la polyligne. Nous pouvons voir concrètement les effets de cela sur la Figure n°17, ci-dessous, et comprendre l'intérêt du rattachement aux coins détectés. Mais, la détection de coins étant somme toutes « rudimentaires », elle manque parfois de précession obligeant à utiliser des distances de rattachement sommets-polygones coins-détectés faibles, afin d'éviter de les rattacher à des coins non pertinents et/ou faux.

Masque brute	Masque simplifié (tolérance 2)	Masque simplifié (tolérance 5)

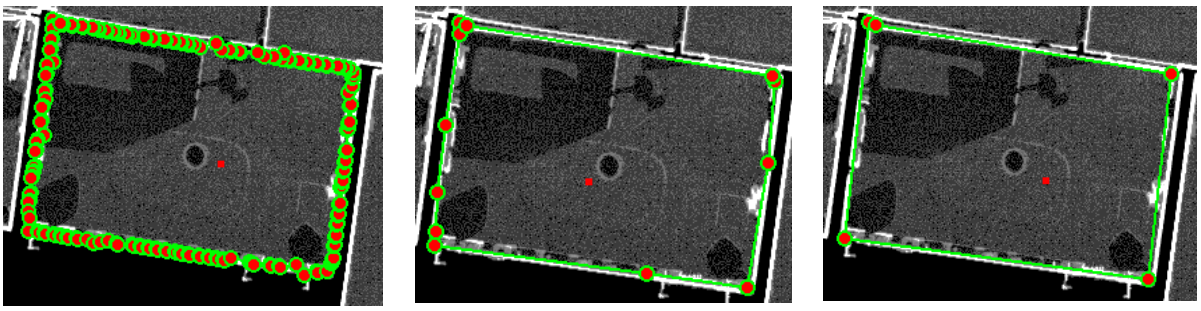


Figure 17 : Impacte de la simplification des masques et de la détection des coins sur la précision des résultats

Quant aux superficies obtenues (tolérance réglée à 3.5) on remarque, pour les pièces tests un écart moyen de 0.3 m² d'écart, entre les superficies prédites et les superficies déterminées (cf. Tableau° 2, ci-dessous), ce qui tout à fait acceptable (écart globalement inferieur au 5% d'écart entre la superficie réelle et calculée, tolérée par la loi Carrée.)

Pieces / Surfaces	**Surface prédite via SAM (en m²)**	**Surface manuellement calculée (en m²)**	**Delta (en m²)**
Bureau	10,8	10,3	0,5
Chambre	9,3	9,7	0,4
Chambre	14,1	14,6	0,3
Chambre	9,5	9,3	0,2
Dégagement	3.8	3,7	0,1
Piece technique	4.2	4,4	0,2
Placard	1.2	1	0.2
Salle de Bain	7,8	7,6	0,2
		Moyenne	0,3

Tableau 2 : Comparaison des superficies prédites et calculées par DAO manuelle

En somme, nous sommes en mesure d'obtenir des résultats exploitables comme l'illustre le tableau n° 2 (tolérance 2) et le tableau n° 9. Cependant ces résultats sont d'une précision inferieure à ceux constatés au TFE Kobena 2023, en moyenne l'écart était de 0.2 m² mètres carrés. De plus, en l'état si de la précision peut être gagnée, ce le sera en dépit de l'esthétique du plan (ligne irrégulière). Cependant, nous obtenons avec cette méthode une définition des limites complètes des pièces permettant d'automatiser notamment le calcul des superficies. De plus, uniquement les sections de polylignes voulues sont dessinées, et cette méthode peut fonctionner sur des intérieurs aux formes peu orthodoxes, pièce non rectangulaire.

Conclusion

Pour conclure, l'automatisation du dessin de plans d'intérieur à partir de nuages de points 3D via le machine/deep learning présente des perspectives intéressantes. L'exploration de solutions comme SAM et RoomFormer a mis en lumière les avantages et les limitations de ces méthodes par rapport aux techniques traditionnelles (RANSAC, transformées de Hough, etc.), résultats plus complets et circonscrits aux dessins des éléments voulus. En contrepartie, nous obtenons une plus faible précision. Ces méthodes nécessitent, en plus de cela des bibliothèques particulières aux compatibilités limitées et demandent des ressources de calcul importante, induisant de plus fortes contraintes matérielles et logicielles que les méthodes dites « classiques ». Enfin, l'utilisation de cartes de densité comme projections 2D des nuages de points simplifie les données tout en conservant les informations essentielles pour la vectorisation. Ces cartes, en améliorant la visibilité des murs et des éléments structuraux, optimisent les résultats des modèles de machine learning et se révèlent prometteuses grâce à leur maniabilité (retouche facile).

Bien que RoomFormer soit peu performant, de nouveaux travaux continuent à être publiés avec des résultats toujours meilleurs, et pourrait à l'avenir être des solutions viables pour un cabinet de géomètre-expert. Quant à SAM, il a montré des résultats prometteurs grâce à ses capacités de segmentation d'images, surtout avec des pré-traitements et post-traitements adaptés. Cependant, les masques générés étant irréguliers, la simplification des contours est nécessaire. Le redressement des angles et la simplification des lignes produisent des polygones plus esthétique, utilisables en DAO, parfois au détriment de la précision. Mais certaines limitations persistent, notamment la gestion des ressources de calcul et la précision des prédictions. En effet, ces dernières subsistent moins précises que les résultats issus des méthodes plus classiques précédemment mentionnées. Mais, plusieurs pistes pourraient être exploitées afin d'améliorer la qualité des prédictions. Notamment l'utilisation/développement d'une méthode plus précise de détection des coins directement dans le nuage de points 3D, qui combinée à la définition globale des pièces par SAM, pourrait offrir des résultats très précis. L'intégration de SAM dans un outil de DAO comme Autocad pourrait également améliorer l'ergonomie et la pertinence de la méthode proposée.

En somme, l'intégration de l'intelligence artificielle, bien qu'aujourd'hui encore peu précise, a un potentiel considérable pour améliorer l'efficacité et la précision des processus de DAO automatisées.

Bibliographie

Travaux universitaires

Talec, Léa. *Mise en place d'un processus de dessin automatisé de plans d'intérieurs à partir de nuages de points acquis par LIDAR*. Mémoire d'ingénieur, Spécialité « Géomètre et Topographe », Le Mans : Cnam ESGT, 2017, 68 p.

Kobena, Kossonou Marc-Emmanuel. *Du cadre juridique de la prise de mesure au scanner 3D à l'automatisation de la production de plans 2D sur nuage de points au sein d'un cabinet de géomètre expert. Mémoire d'ingénieur, Spécialité « Géomètre et Topographe », Le Mans : Cnam ESGT, 2023, 61 p.

Articles de périodiques électroniques

Chen, Jiacheng, Yiming Qian, and Yasutaka Furukawa. "HEAT : Holistic Edge Attention Transformer for Structured Reconstruction." *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, (consulté le 03/06/2024).

Hershberger, John, and Jack Snoeyink. "Speeding Up the Douglas-Peucker Line-Simplification Algorithm." *DEC Systems Research Center*, 130 Lytton Ave, Palo Alto, CA 94305 USA, (consulté le 25/06/2024). Disponible sur : https://www.cs.ubc.ca/sites/default/files/tr/1992/TR-92-07_0.pdf.

Stekovic, Sinisa, et al. "MonteFloor : Extending MCTS for Reconstructing Accurate Large-Scale Floor Plans." *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, (consulté le 02/07/2024).

Talotta, Anselmo, Valentin Radu, and Lorenzo Sorgi. "Floorplan Generation from Noisy Point Cloud." *INTech, Amazon*, (consulté le 02/07/2024). Disponible sur https://www.amazon.science/publications/floorplan-generation-from-noisy-point-cloud :

Yue, Yuanwen, et al. "Connecting the Dots : Floorplan Reconstruction Using Two-Level Queries." *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. (Consulté le 04/01/2024). Disponible sur : [RoomFormer](https://ywyue.github.io/RoomFormer/).

Kirillov, Alexander, et al. "Segment Anything." *arXiv preprint arXiv :2304.02643*, 2023. (Consulté le 22/11/2023).

Kirillov, Alexander, et al. "Segment Anything : A Large-scale Dataset and Model for Segmentation." *arXiv preprint arXiv :2304.02643*, 2023. (Consulté le 22/01/2024).

Table des annexes

```python
import sys
import os
import cv2
import numpy as np
import laspy
import open3d as o3d
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QLabel, QPushButton, QVBoxLayout,
    QHBoxLayout, QWidget, QFileDialog, QListWidget, QScrollArea,
    QSplitter, QShortcut, QDialog, QFormLayout, QLineEdit,
    QDialogButtonBox, QMessageBox
)
from PyQt5.QtGui import QPixmap, QPainter, QPen, QImage, QKeySequence, QCursor
from PyQt5.QtCore import Qt, QPoint
from ultralytics import SAM
from skimage.measure import approximate_polygon
from shapely.geometry import Polygon
import ezdxf
import traceback
```

Annexe 1 : Importation des bibliothèques (Pour l'interface de l'implémentation de SAM)

```python
class ImageLabel(QLabel):
    def __init__(self, params):
        super().__init__()
        self.params = params
        self.image = None
        self.points = []
        self.polygons = []
        self.altitude_points = []
        self.delta_texts = []
        self.simplified = []
        self.selected_polygon_index = None
        self.selected_point_index = None
        self.history = []
        self.device = "gpu"
        self.pixel_to_point_mapping = None
        self.lowest_points = None
        self.delta_heights = None
        self.mean_points = None
        self.current_action = None
        self.load_model()

    def load_model(self):
        try:
            self.model = SAM(self.params['model_path'])
            self.model.info()
        except Exception as e:
            QMessageBox.critical(self, "Erreur de Chargement du Modèle", str(e))

    def setImage(self, pixmap):
        self.image = self.pixmap_to_cv(pixmap)
        self.reset_annotations()
        self.setFixedSize(pixmap.size())
        self.repaint()

    def set_pixel_to_point_mapping(self, mapping):
        self.pixel_to_point_mapping = mapping['unique_coordinates']
        self.lowest_points = mapping['lowest_points']
        self.delta_heights = mapping['delta_heights']
        self.mean_points = mapping['unique_coordinates']

    def paintEvent(self, event):
        super().paintEvent(event)
        if self.image is not None:
            painter = QPainter(self)
            painter.drawPixmap(self.rect(), self.cv_to_pixmap(self.image))
            self.draw_points(painter)
            self.draw_polygons(painter)

    def mousePressEvent(self, event):
        if event.button() == Qt.LeftButton:
            self.handle_left_click(event)
        elif event.button() == Qt.RightButton:
            self.handle_right_click(event)

    def mouseMoveEvent(self, event):
        if self.selected_polygon_index is not None and self.selected_point_index is not None:
            self.polygons[self.selected_polygon_index][self.selected_point_index] = [event.pos().x(), event.pos().y(
            self.repaint()
```

Annexe 2: Gestion des évènements et des affichages de l'interface pour l'implémentation du modèle SAM, n°1

```
def handle_left_click(self, event):
    pos = event.pos()
    if self.selected_polygon_index is not None and self.selected_point_index is not None:
        self.save_history()
        self.polygons[self.selected_polygon_index][self.selected_point_index] = [pos.x(), pos.y()]
        self.selected_polygon_index = None
        self.selected_point_index = None
    else:
        if not self.select_existing_point(pos):
            self.points.append(pos)
            if self.current_action == "altitude":
                self.add_altitude_point(pos)
            elif self.current_action == "delta":
                self.add_delta_text(pos)
            elif self.current_action == "polygon":
                self.update_mask()
    self.repaint()

def handle_right_click(self, event):
    self.points.append(event.pos())
    self.process_all_masks()
    self.repaint()

def select_existing_point(self, pos):
    for i, polygon in enumerate(self.polygons):
        for j, point in enumerate(polygon):
            if self.is_near_point(pos, point):
                self.selected_polygon_index = i
                self.selected_point_index = j
                return True
    return False

def reset_annotations(self):
    self.points = []
    self.polygons = []
    self.altitude_points = []
    self.delta_texts = []

def draw_points(self, painter):
    pen = QPen(Qt.red, 5)
    painter.setPen(pen)
    for point in self.points:
        painter.drawPoint(point)
        for altitude_point, altitude in self.altitude_points:
            if point == altitude_point:
                painter.drawText(point.x() + 5, point.y() - 5, f'z = {altitude:.2f}')
        for delta_point, delta in self.delta_texts:
            if point == delta_point:
                painter.drawText(point.x() + 5, point.y() - 5, f'H = {delta:.2f}')
```

Annexe 3: Gestion des évènements et des affichages de l'interface pour l'implémentation du modèle SAM, n°2

```
def draw_polygons(self, painter):
    if self.polygons:
        pen = QPen(Qt.green, 2)
        painter.setPen(pen)
        for polygon in self.polygons:
            for i in range(len(polygon) - 1):
                painter.drawLine(QPoint(*polygon[i]), QPoint(*polygon[i + 1]))
            painter.drawLine(QPoint(*polygon[-1]), QPoint(*polygon[0]))  # Fermer le polygone
            self.draw_polygon_vertices(painter, polygon)

def draw_polygon_vertices(self, painter, polygon):
    painter.setBrush(Qt.red)
    for point in polygon:
        painter.drawEllipse(QPoint(point[0], point[1]), 5, 5)

def undo_last_point(self):
    if self.points:
        self.save_history()
        self.points.pop()
        self.update_mask()
        self.repaint()
```

Annexe 4: Gestion des évènements et des affichages de l'interface pour l'implémentation du modèle SAM, n°3

```
def update_mask(self):
    if self.image is not None and self.points:
        temp_image_path = "temp_image.png"
        cv2.imwrite(temp_image_path, self.image)

        # Détection des coins
        dcorners = self.corner_detection(temp_image_path)

        # Utilisation du premier point comme point d'invite
        input_points = [[p.x(), p.y()] for p in self.points]
        ann = self.model.predict(temp_image_path, points=input_points[0], labels=[1])

        # Approximation du polygone basé sur l'annotation
        self.simplified = approximate_polygon(ann[0].masks.xy[0], tolerance=5)
        self.simplified = self.adjust_polygon_angles(self.simplified)

        # Attacher les points les plus proches détectés
        self.attach_closest_points(dcorners)

        # Vérifier si le polygone est valide et l'ajouter à la liste
        if self.is_valid_polygon(self.simplified):
            self.polygons.append(self.simplified)

        # Supprimer le premier point de la liste des points après la génération du polygone
        self.points.pop(0)

        # Nettoyage de l'image temporaire
        os.remove(temp_image_path)

        # Réinitialiser l'état et mettre à jour l'affichage
        self.selected_polygon_index = None
        self.selected_point_index = None
        self.repaint()
```

Annexe 5: Fonction d'exécution du modèle SAM, par invite (clique en un point), et des post-traitements associés et mise à jour de l'affichage

```
def process_all_masks(self):
    if self.image is not None and self.points:
        temp_image_path = "temp_every_image.png"
        cv2.imwrite(temp_image_path, self.image)
        dcorners = self.corner_detection(temp_image_path)
        results = self.model(temp_image_path)
        for k in results[0].masks.xy:
            self.simplified = approximate_polygon(np.array(k), tolerance=5)
            self.simplified = self.adjust_polygon_angles(self.simplified)
            self.attach_closest_points(dcorners)
            if self.is_valid_polygon(self.simplified):
                self.polygons.append(self.simplified)
        os.remove(temp_image_path)
```

Annexe 6: Fonction d'exécution du modèle SAM, automatique (génération d'une grille de points)), et des post-traitements associés et mise à jour de l'affichage

```python
def adjust_polygon_angles(self, polygon, tolerance=None):
    if tolerance is None:
        tolerance = self.params['tolerance']
    adjusted_polygon = polygon.copy()
    n = len(polygon)
    for i in range(n):
        a = np.array(polygon[i - 1])
        b = np.array(polygon[i])
        c = np.array(polygon[(i + 1) % n])
        angle = self.calculate_angle(a, b, c)
        if abs(angle - 90) <= tolerance:
            adjusted_polygon[i] = self.adjust_angle(b, a, c, angle)
    return adjusted_polygon

def adjust_angle(self, b, a, c, angle):
    ba = a - b
    bc = c - b
    ba_len = np.linalg.norm(ba)
    bc_len = np.linalg.norm(bc)
    ba_norm = ba / ba_len
    bc_norm = bc / bc_len
    if ba_len < bc_len:
        move_vector = (ba_norm + np.array([-ba_norm[1], ba_norm[0]])) * (90 - angle) / 180
    else:
        move_vector = (bc_norm + np.array([-bc_norm[1], bc_norm[0]])) * (90 - angle) / 180
    return (b + move_vector * min(ba_len, bc_len) / 10).tolist()

def calculate_angle(self, a, b, c):
    ab = np.array([b[0] - a[0], b[1] - a[1]])
    bc = np.array([c[0] - b[0], c[1] - b[1]])
    ab_norm = ab / np.linalg.norm(ab)
    bc_norm = bc / np.linalg.norm(bc)
    cos_angle = np.dot(ab_norm, bc_norm)
    angle = np.arccos(np.clip(cos_angle, -1.0, 1.0))
    return np.degrees(angle)

def corner_detection(self, image_path):
    gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    dcorners = cv2.goodFeaturesToTrack(gray, maxCorners=1000, qualityLevel=0.2, minDistance=10)
    return np.int0(dcorners).reshape(-1, 2)

def attach_closest_points(self, dcorners, distance_threshold=None):
    if distance_threshold is None:
        distance_threshold = self.params['pixel_threshold']
    for vert_idx, vertex in enumerate(self.simplified):
        distances = np.linalg.norm(dcorners - vertex, axis=1)
        closest_point_idx = np.argmin(distances)
        if distances[closest_point_idx] <= distance_threshold:
            self.simplified[vert_idx] = dcorners[closest_point_idx]
```

```python
def is_valid_polygon(self, new_polygon):
    new_poly = Polygon(new_polygon)
    min_area = 50 ** 2
    if new_poly.area < min_area:
        return False
    for existing_polygon in self.polygons:
        existing_poly = Polygon(existing_polygon)
        if existing_poly.within(new_poly) or new_poly.within(existing_poly):
            return False
    return True
```

Annexe 7: Algorithme de post-traitement

```python
def save_polygon_to_dxf(self, polygon_points, filename):
    try:
        doc = ezdxf.new()
        msp = doc.modelspace()

        if self.pixel_to_point_mapping is None:
            raise ValueError("Le mapping pixel-point n'est pas chargé")

        unique_coords = self.pixel_to_point_mapping
        mean_points = self.mean_points

        for polygon in polygon_points:
            polygon = polygon.tolist()

            if polygon[0] != polygon[-1]:
                polygon.append(polygon[0])

            transformed_polygon = self.transform_polygon(polygon, unique_coords, mean_points)
            msp.add_lwpolyline(transformed_polygon)

        for point, altitude in self.altitude_points:
            index = self.find_closest_index(point, unique_coords)
            georef_coord = mean_points[index]
            msp.add_point(georef_coord[:2], dxfattribs={'layer': 'AltitudePoints', 'color': 1})
            text = msp.add_text(f'z = {altitude:.2f}', dxfattribs={'height': self.params['dxf_text_height'], 'la
            text.set_dxf_attrib('insert', (float(georef_coord[0]) + self.params['dxf_text_offset_x'], float(geor

        for point, delta in self.delta_texts:
            index = self.find_closest_index(point, unique_coords)
            georef_coord = mean_points[index]
            text = msp.add_text(f'H = {delta:.2f}', dxfattribs={'height': self.params['dxf_text_height'], 'layer
            text.set_dxf_attrib('insert', (float(georef_coord[0]), float(georef_coord[1])))

        doc.saveas(filename)
        print(f"Fichier DXF sauvegardé avec succès sous {filename}")

    except Exception as e:
        print(f"Erreur lors de l'exportation en DXF: {e}")
        traceback.print_exc()
```

Annexe 8: Fonction d'exports des éléments générés dans un fichier au format DXF

```python
def find_closest_index(self, point, unique_coords):
    distances = np.linalg.norm(unique_coords - np.array([point.x(), point.y()]), axis=1)
    return np.argmin(distances)

def transform_polygon(self, polygon, unique_coords, mean_points):
    transformed_polygon = []
    for point in polygon:
        index = self.find_closest_index(QPoint(point[0], point[1]), unique_coords)
        georef_coord = mean_points[index][:2]
        transformed_polygon.append(georef_coord)
    return transformed_polygon

def get_points(self):
    return self.points

def is_near_point(self, pos, point, threshold=10):
    return np.linalg.norm(np.array([pos.x() - point[0], pos.y() - point[1]])) < threshold

def remove_polygon(self):
    if self.selected_polygon_index is not None:
        self.save_history()
        self.polygons.pop(self.selected_polygon_index)
        self.selected_polygon_index = None
        self.repaint()

def save_history(self):
    self.history.append((self.polygons.copy(), self.points.copy()))

def undo(self):
    if self.history:
        self.polygons, self.points = self.history.pop()
        self.repaint()
```

Annexe 9; Gestion des éléments du fichiers de coordonnées npy et des retours en arrière

```
def add_altitude_point(self, pos):
    try:
        if self.pixel_to_point_mapping is not None:
            if self.lowest_points is None or len(self.lowest_points) == 0:
                print("Pas de données de points les plus bas disponibles.")
                return

            unique_coords, lowest_points = self.pixel_to_point_mapping, self.lowest_points
            clicked_coord = np.array([pos.x(), pos.y()])

            distances = np.linalg.norm(unique_coords - clicked_coord, axis=1)
            closest_index = np.argmin(distances)

            if distances[closest_index] < self.params['pixel_threshold']:
                altitude = lowest_points[closest_index][2]
                self.altitude_points.append((pos, altitude))
                print(f"Point d'altitude ajouté aux coordonnées de l'interface {pos} avec une altitude de {altit
            else:
                print(f"Aucun point trouvé dans la distance seuil. Distance la plus proche : {distances[closest_
        else:
            print("Le mapping pixel-point est nul.")
    except Exception as e:
        print(f"Erreur dans add_altitude_point: {e}")
```

Annexe 10: Gestion des altimétries du fichiers de coordonnées npy

```
def add_delta_text(self, pos):
    try:
        if self.pixel_to_point_mapping is not None:
            if self.delta_heights is None or len(self.delta_heights) == 0:
                print("Pas de données sur les hauteurs delta disponibles.")
                return

            unique_coords, delta_heights = self.pixel_to_point_mapping, self.delta_heights
            clicked_coord = np.array([pos.x(), pos.y()])

            distances = np.linalg.norm(unique_coords - clicked_coord, axis=1)
            closest_index = np.argmin(distances)

            if distances[closest_index] < self.params['pixel_threshold']:
                delta_h = delta_heights[closest_index]
                self.delta_texts.append((pos, delta_h))
                print(f"Texte delta ajouté aux coordonnées de l'interface {pos} avec une delta de {delta_h}")
            else:
                print(f"Aucune delta trouvée dans la distance seuil. Distance la plus proche : {distances[closes
        else:
            print("Le mapping pixel-point est nul.")
    except Exception as e:
        print(f"Erreur dans add_delta_text: {e}")
```

Annexe 11: Gestion et affichage des textes de hauteur, (lues depuis le fichier de coordonnées npy

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Outil d'Annotation d'Image")
        self.setGeometry(100, 100, 1000, 800)
        self.params = {
            'model_path': r"D:\TFE\FastSAM\Weights\sam_l.pt",
            'tolerance': 30,
            'dxf_text_height': 0.13,
            'dxf_text_offset_x': 0.1,
            'dxf_text_offset_y': 0.1,
            'pixel_threshold': 20
        }
        self.init_ui()

    def init_ui(self):
        self.imageLabel = ImageLabel(self.params)
        self.imageLabel.setAlignment(Qt.AlignCenter)
        self.scrollArea = QScrollArea()
        self.scrollArea.setWidget(self.imageLabel)
        self.scrollArea.setAlignment(Qt.AlignCenter)
        self.settingsButton = QPushButton("Paramètres")
        self.settingsButton.clicked.connect(self.open_settings_dialog)
        self.generateDensityButton = QPushButton("Générer la Carte de Densité")
        self.generateDensityButton.clicked.connect(self.open_density_map_dialog)
        self.saveDxfButton = QPushButton("Enregistrer en DXF")
        self.saveDxfButton.clicked.connect(self.save_dxf)
        self.loadImageButton = QPushButton("Charger une Image")
        self.loadImageButton.clicked.connect(self.load_image)
        self.loadMappingButton = QPushButton("Charger le Mapping")
        self.loadMappingButton.clicked.connect(self.load_mapping)
        self.addPolygonButton = QPushButton("Ajouter un Polygone")
        self.addPolygonButton.clicked.connect(self.add_polygon)
        self.addPointButton = QPushButton("Ajouter un Point d'Altitude au DXF")
        self.addPointButton.clicked.connect(self.add_altitude_point)
        self.addTextButton = QPushButton("Ajouter un Texte Delta au DXF")
        self.addTextButton.clicked.connect(self.add_delta_text)
        self.undoButton = QPushButton("Annuler le Dernier Point")
        self.undoButton.clicked.connect(self.imageLabel.undo)
        self.validateButton = QPushButton("Valider les Points")
        self.validateButton.clicked.connect(self.validate_points)
        self.removePolygonButton = QPushButton("Supprimer le Polygone")
        self.removePolygonButton.clicked.connect(self.imageLabel.remove_polygon)
        self.cancelButton = QPushButton("Annuler")
        self.cancelButton.clicked.connect(self.cancel)
        self.imageListWidget = QListWidget()
        self.imageListWidget.currentItemChanged.connect(self.change_image)
        self.undoShortcut = QShortcut(QKeySequence("Ctrl+Z"), self)
        self.undoShortcut.activated.connect(self.imageLabel.undo)
        self.layout_ui()
```

Annexe 12: Paramétrage de la fenêtre de l'interface de l'implémentation du modèle SAM

```
def layout_ui(self):
    topButtonLayout = QHBoxLayout()
    topButtonLayout.addWidget(self.settingsButton)
    topButtonLayout.addWidget(self.generateDensityButton)
    topButtonLayout.addWidget(self.saveDxfButton)
    buttonLayout = QHBoxLayout()
    buttonLayout.addWidget(self.loadImageButton)
    buttonLayout.addWidget(self.loadMappingButton)
    buttonLayout.addWidget(self.addPolygonButton)
    buttonLayout.addWidget(self.addPointButton)
    buttonLayout.addWidget(self.addTextButton)
    buttonLayout.addWidget(self.undoButton)
    buttonLayout.addWidget(self.validateButton)
    buttonLayout.addWidget(self.removePolygonButton)
    buttonLayout.addWidget(self.cancelButton)
    mainLayout = QVBoxLayout()
    mainLayout.addLayout(topButtonLayout)
    splitter = QSplitter(Qt.Vertical)
    splitter.addWidget(self.scrollArea)
    splitter.addWidget(self.imageListWidget)
    mainLayout.addWidget(splitter)
    mainLayout.addLayout(buttonLayout)
    container = QWidget()
    container.setLayout(mainLayout)
    self.setCentralWidget(container)

def open_density_map_dialog(self):
    dialog = DensityMapDialog()
    if dialog.exec_():
        data_file, output_folder, width, height = dialog.get_params()
        self.generate_density_map(data_file, output_folder, width, height)

def load_image(self):
    options = QFileDialog.Options()
    options |= QFileDialog.ReadOnly
    fileName, _ = QFileDialog.getOpenFileName(self, "Ouvrir le Fichier Image", "", "Images (*.png *.xpm *.jpg);;
    if fileName:
        self.imageListWidget.addItem(fileName)

def load_mapping(self):
    options = QFileDialog.Options()
    options |= QFileDialog.ReadOnly
    fileName, _ = QFileDialog.getOpenFileName(self, "Ouvrir le Fichier de Mapping", "", "Fichiers Numpy (*.npy);
    if fileName:
        mapping_data = np.load(fileName, allow_pickle=True).item()
        self.imageLabel.set_pixel_to_point_mapping(mapping_data)
```

Annexe 13: Paramétrage et gestion des évènements de la fenêtre de l'interface de l'implémentation du modèle SAM

```
def open_settings_dialog(self):
    dialog = SettingsDialog(self)
    if dialog.exec_():
        self.params = dialog.get_params()
        self.imageLabel.params = self.params


class SettingsDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Configuration des Paramètres")
        self.init_ui()

    def init_ui(self):
        self.modelPathEdit = QLineEdit(r"D:\TFE\FastSAM\Weights\sam_l.pt", self)
        self.toleranceEdit = QLineEdit("30", self)
        self.dxfTextHeightEdit = QLineEdit("0.13", self)
        self.dxfTextOffsetXEdit = QLineEdit("0.1", self)
        self.dxfTextOffsetYEdit = QLineEdit("0.1", self)
        self.pixelThresholdEdit = QLineEdit("20", self)

        layout = QFormLayout()
        layout.addRow("Chemin du modèle:", self.modelPathEdit)
        layout.addRow("Tolérance pour les angles:", self.toleranceEdit)
        layout.addRow("Hauteur du texte DXF:", self.dxfTextHeightEdit)
        layout.addRow("Décalage X du texte DXF:", self.dxfTextOffsetXEdit)
        layout.addRow("Décalage Y du texte DXF:", self.dxfTextOffsetYEdit)
        layout.addRow("Seuil en pixels:", self.pixelThresholdEdit)

        self.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)
        layout.addWidget(self.buttonBox)

        self.setLayout(layout)

    def get_params(self):
        return {
            'model_path': self.modelPathEdit.text(),
            'tolerance': int(self.toleranceEdit.text()),
            'dxf_text_height': float(self.dxfTextHeightEdit.text()),
            'dxf_text_offset_x': float(self.dxfTextOffsetXEdit.text()),
            'dxf_text_offset_y': float(self.dxfTextOffsetYEdit.text()),
            'pixel_threshold': int(self.pixelThresholdEdit.text())
        }
```

Annexe 14: Gestion du paramétrage de variables via l'interface graphique de l'implémentation du modèle SAM

```
def load_image_from_path(self, path):
    if path:
        pixmap = QPixmap(path)
        self.imageLabel.setImage(pixmap)
        self.scrollArea.setWidgetResizable(True)

def change_image(self, current, previous):
    if current:
        pixmap = QPixmap(current.text())
        self.imageLabel.setImage(pixmap)
        self.scrollArea.setWidgetResizable(True)

def validate_points(self):
    points = self.imageLabel.get_points()
    print("Points validés :", points)

def save_dxf(self):
    if self.imageLabel.polygons or self.imageLabel.altitude_points or self.imageLabel.delta_texts:
        options = QFileDialog.Options()
        fileName, _ = QFileDialog.getSaveFileName(self, "Enregistrer le Fichier DXF", "", "Fichiers DXF (*.dxf);
        if fileName:
            self.imageLabel.save_polygon_to_dxf(self.imageLabel.polygons, fileName)

def cancel(self):
    self.imageLabel.setImage(None)
    self.imageListWidget.clear()

def add_altitude_point(self):
    self.imageLabel.current_action = "altitude"

def add_delta_text(self):
    self.imageLabel.current_action = "delta"

def add_polygon(self):
    self.imageLabel.current_action = "polygon"
```

Annexe 15: Paramétrage et gestion des évènements de la fenêtre de l'interface de l'implémentation du modèle SAM

```python
def generate_density_map(self, data_file, output_folder, width, height):
    ply_path = os.path.join(output_folder, os.path.splitext(os.path.basename(data_file))[0] + '.ply')
    mapping_file = os.path.join(output_folder, 'pixel_to_point_mapping.npy')
    density_file = os.path.join(output_folder, 'density_map.png')

    # Lire les points du fichier LAS
    points = read_las_file(data_file)

    # Initialiser les matrices de densité, altitude minimale et maximale, et delta_heights
    density = np.zeros((height, width))
    min_altitude = np.full((height, width), np.inf)  # Altitude minimale initialisée à +infini
    max_altitude = np.full((height, width), -np.inf)  # Altitude maximale initialisée à -infini
    delta_heights = np.zeros((height, width))  # Pour les deltas

    for point in points:
        x, y, z = point  # Supposons que point est (x, y, z)
        ix, iy = int(x), int(y)  # Convertir en indices de la matrice

        # Mettre à jour la densité
        density[iy, ix] += 1

        # Mettre à jour l'altitude minimale et maximale
        if z < min_altitude[iy, ix]:
            min_altitude[iy, ix] = z
        if z > max_altitude[iy, ix]:
            max_altitude[iy, ix] = z

        # Calculer le delta entre l'altitude max et min
        delta_heights[iy, ix] = max_altitude[iy, ix] - min_altitude[iy, ix]

    # Sauvegarder le mapping avec les valeurs correctes
    np.save(mapping_file, {
        "unique_coordinates": density,
        "lowest_points": min_altitude,
        "delta_heights": delta_heights
    })

    # Ecriture des fichiers PLY et image de densité
    write_ply_file(points, ply_path)
    export_density(density, output_folder, 'density_map')

    # Charger l'image générée et mettre à jour l'interface utilisateur
    self.load_image_from_path(density_file)
    self.imageLabel.set_pixel_to_point_mapping(np.load(mapping_file, allow_pickle=True).item())
```

Annexe 16: Fonction de génération des cartes de densités

Liste des figures

Liste des tableaux

Optimisation de la chaîne de production de plans d'intérieur depuis des nuages de points 3D

Mémoire d'Ingénieur Le CNAM ESGT., Le Mans 2024

RESUMÉ

Ce mémoire explore les solutions d'optimisation de la chaîne de production de plans d'intérieur à partir de nuages de points 3D. L'utilisation de la technologie 3D LIDAR a transformé a rendu travail sur le terrain plus rapide et plus précis, mais augmentant également la complexité du traitement des données.

Le mémoire étudie deux approches principales pour automatiser la vectorisation des nuages de points en plans 2D : l'utilisation du modèle RoomFormer pour la reconstruction des plans d'étage à partir de cartes de densité (représentation 2D issue du nuage de points) et le Segment Anything Model (SAM), un modèle de segmentation d'images, implémenté pour segmenter les pièces des cartes de densité précédemment mentionnées. Les résultats montrent que si RoomFormer est limité, le modèle SAM, associé à des prétraitements et post-traitements spécifiques des données et des sorties, constitue une solution prometteuse.

Mots clés : Nuages de points 3D, LIDAR, Vectorisation, RoomFormer, Reconstruction de plans d'étage, Cartes de densité, Segment Anything Model (SAM), Segmentation d'images

---

SUMMARY

This thesis explores optimization solutions for the production chain of interior plans from 3D point clouds. The use of 3D LIDAR technology has made fieldwork faster and more accurate, but has also increased the complexity of data processing.

The thesis examines two main approaches to automate the vectorization of point clouds into 2D plans: the use of the RoomFormer model for floor plan reconstruction from density maps (a 2D representation derived from the point cloud) and the Segment Anything Model (SAM), an image segmentation model implemented to segment the rooms in the previously mentioned density maps. The results show that while RoomFormer has limitations, the SAM model, combined with specific data preprocessing and postprocessing techniques, represents a promising solution.

Key words: 3D point clouds, LIDAR, Vectorization, RoomFormer, Floor plan reconstruction, Density maps, Segment Anything Model (SAM), Image segmentation