Rapport de stage Master2 recherche

Benoît Petitpas <petitpas@enst.fr>

12 septembre 2008

er

tt

tt

tt

tt

Remerciements

Je tiens à remercier très chaleureusement Michel Roux, mon maître de stage et maître de conférence à TELECOM ParisTech, pour toute l'aide qu'il m'a apporté. Ces conseils et nos discutions autour des algorithmes et méthodes à utiliser m'ont énormément motivé et appris.

Je remercie aussi toute l'équipe encadrante du master 2 SIG pour l'intérêt qu'ils ont montré pour mes travaux et tout particulièrement M.Olivier de Joinville pour sa patience alors qu'il me supervisait durant mon stage, ainsi que Jean-Paul Rudant devenu mon directeur de thèse.

Je remercie ensuite tous les stagiaires de TELECOM ParisTech, Charles-Alban Deledalle, Jean-Christophe Perles, Yajin Yan, Renaud Fallourd, pour leur bonne humeur, leur chaleur ainsi que leurs conseils avisés dans certains domaines.

Enfin je tiens à remercier ma famille pour le gros travail de relecture et de correction orthographique que je leur ai imposés.

Résumé

La 3d urbaine est un sujet sur lequel travaille de nombreux chercheurs depuis plus de 10 ans et a fait de nombreux progrès ces dernières années. L'intérêt du public pour ces technologies est croissant s'il on en croit les résultats de Google earth, Google maps ou le GéoPortail. La recherche dans ce domaine a été aidée par les campagnes effectuées par l'IGN[1], les résultats obtenus sur la ville d'Amiens sont d'excellente qualité. Ces campagnes ont été réalisées avec l'aide du département TSI de l'institut TELECOM Paristech[2].

La genèse du stage se trouve dans le projet Terranumérica[3], qui a pour but d'adapter différentes technologies de reconstruction 3D urbaines pour les intégrer à un seul et même système, pour des applications aussi diverses que l'urbanisme,la culture (patrimoine, arts, loisirs et jeux), l'éducation, le tourisme, la sécurité civile ou tout simplement le citoyen dans la ville. Il pourra être consulté sur diverses plate-formes comme l'Internet, le téléphone portable, les GPS auto, des lunettes d'observation ou des salles immersives.

Dans le cadre de ce projet, nous avons cherché à utiliser deux technologies conjointement, la technologie LIDAR[4] et les photos hautes résolutions. Le stage tourna autour de deux axes bien distincts qui sont l'inpainting des images, ou complétion des occlusions et l'inpainting des données 3d, là ou le mobilier urbain provoque une ombre sur la façade.
Dans l'inpainting de images nous nous servons d'algorithmes existants que nous avons ensuite spécialisés au problème des images de façades.
Dans l'inpainting 3d nous tentons de créer des points 3D à partir des points LIDAR et de l'image segmentée. On cherche, pour cela, le ou les meilleurs plans approximant les données. On projette alors tous les pixels sur ces plans.
Le fait de pouvoir mettre toutes les données dans la même géométrie nous permet d'interagir entre les données images et 3D.
Le résultat est une automatisation des occlusions et de l'inpainting des images qui gardent les structures et la texture, ainsi que la création d'une carte de profondeur qui permettant la complétions des données 3D.

[1] http ://www.ign.fr/
[2] http ://www.telecom-paristech.fr/recherche/traitement-signal-image/
[3] http ://terranumerica.com/
[4] LIght Detection And Ranging

Abstract

The urban 3D is an interesting subject which is study by many researchers for 10 years and have made huge progress during the last years. The interest for such a technology is increasing, considering the results of Google earth, Google maps, or GeoPortail. The research was helped by the 3D campains of the national geographical institut, IGN, and the results on Amiens are trully exceptional. This campains was realized with TELECOM Paristech's TSI laboratory.

The internship genesis come from Terranumerica's project, wich aims to adapt differents technologies of urban 3D reconstruction to integrate them in one single system for miscalaneous applications as urbanism, culture (patrimonia, arts, ...), education, tourism, civil security or helping the city citizen. This system will be implemented in differents frameworks as Internet, mobile phone, GPS observation glasses or immersive rooms.

For this project, we search to use two technologies at the same time : the LIDAR and high resolution images. The intership will be axed on two subjects : the image inpainting and 3d inpainting, where urban furnitures are shadowing the facade.
In the image inpainting we are using existing algorithms wich we are specifing to the facades images problem.
In 3D inpainting, we are trying to create 3D points from LIDAR points and segmentation of the image. For that we are searching le best fitting plans. Then we project all the pixels on this plans.
We can put all the data in the same geometry and let us interact between image and 3D data.
As a result, we can create automaticaly the occlusions and inpaint them keeping structures and textures, as well as the creation of a depth map wich let us completing the data.

Table des matières

Liste des tableaux

Table des figures

Glossaire et sigles utiles

LIDAR	LIght Detection And Ranging
TSI	Traitement du Signal et des Images
IGN	Institut Géographique National
HR	Haute Résolution
XML	Extensible Markup Language
RANSAC	RANdom SAmple Consensus
GSL	GNU Scientific Library
ICM	Iterated Conditional Modes
IMG	abréviation pour “image”

Introduction

Des millions de gens ont vu tomber une pomme, Newton est le seul qui se soit demandé pourquoi.[Bernard Baruch]

Depuis bientôt 10 ans, la 3D urbaine fait de grands progrès notamment sous l'impulsion de nouvelles technologies internet comme "Google earth", "Terraexplorer" ou "NASA World Wind". De nombreux laboratoires de recherche français ont de même travaillé sur le sujet comme le département TII du laboratoire GET/TSI de l'école TELECOM ParisTech,dans lequel mon stage fut effectué ou le laboratoire MATIS de l'IGN partenaire du master 2 SIG. Ces nouvelles technologies intéressent certain industriels qui voient, dans l'intérêt que porte le publique à la 3D urbaine, de nombreuses possibilités de déclinaisons commerciales. Ainsi Thalès, grand groupe français s'est investi dans un projet visant à rassembler un maximum de connaissances dans le domaine de la 3D urbaine. Ce projet se nomme Terranumerica. Parmi les collaborateurs du projet on retrouve notamment le laboratoire GET/TSI. Dans ce projet nous nous plaçons dans la partie traitement des données images et LIDAR.

Les donnés sont acquises par un système développé par l'IGN permettant la prise au sol de photos hautes résolutions, ainsi que de points LIDAR. Ces données sont toutes géoréférencées, rendant la mise en correspondance des données plus aisée. Mais si l'on applique directement les images comme texture sur les points 3D maillés par une triangulation, on risque de voir apparaître de nombreux éléments indésirables : piétons, voitures ou matériels urbains. De plus, la densité des points 3D ne permet pas une bonne approche de la texture, et les erreurs inhérentes au LIDAR et au système de prise de vue (couplage GPS[5]/INS[6]) provoquent une mise en géométrie 3D de mauvaise qualité.

Ainsi le travail confié au laboratoire est double, dans un premier temps nous expliquerons comment nous avons traité les images, pour faire disparaître les éléments indésirables que ce soit à partir de l'image uniquement ou à l'aide des points 3D. Pour effectuer cette tâche, nous nous sommes appuyés sur différentes techniques existantes expliquées en 1.1 que nous avons spécialisé à notre problème par un ajout d'innovations expliquées en 1.2, qui ont rendues le système beaucoup plus robuste. Nous avons aussi travaillé sur la mise au point automatique de masque d'inpainting, c'est à dire qu'à partir du résultat d'un filtrage 3D, nous nous servons de ce qui n'est pas considéré comme façade pour supprimer sur l'image les parties non façades et permettre ainsi un inpainting automatique.

Dans un second temps nous avons travaillé à la densification du nuage de points 3D, car nous manquions d'informations pour la mise en place d'une carte de profondeur robuste. Ce travail fut rendu robuste par l'utilisation de l'information contenue dans les images. En effet nous expliquerons dans la partie 2 comment nous nous sommes servi de l'information de segmentation des images pour permettre une densification efficace des points 3D. De plus une fois le nuage de points 3D rendu dense, on inpaintera les parties où les points 3D sont inexistants grâce à la construction d'une carte de profondeur. En effet en LIDAR chaque obstacle créer une "ombre" sur la façade.
Ainsi nous avons produit deux applications permettant de transformer 2 donnés incomplètes, des images avec du mobilier urbain et des données 3D peux denses en un nuage de point où il y a autant

[5]GPS : Global Positionning System
[6]Inertial Navigation System

de points que de pixel ainsi que des images de façades uniquement.

Chapitre 1

Inpainting image

Introduction

Dans ce chapitre nous nous intéressons au problème délicat de l'inpainting sur les images. Nous nous y intéresserons dans le cadre du projet Terranumerica qui a pour but de reconstruire des façades urbaines à l'aide de nuages de points 3D et d'images haute définition. Or sur ces dernières apparaissent toujours, mobiliers urbains, piétons, voitures ou tout autre élément gênant une vision complète de la façade. Ainsi s'il on considère le travail de filtrage comme ayant été effectué, il reste sur l'image des parties vides appelées occlusions qui sont à l'emplacement des éléments gênants. Un soin important fut apporté à la mise en place des technologies permettant la désocclusion ou "inpainting" des parties supprimées. Nous nous sommes basés, pour ce travail, sur des recherches que nous expliquerons en 1.1 et sur lesquelles nous avons apporté des innovations permettant de répondre plus spécifiquement au problème des images de façades définies dans la partie 1.2.

1.1 Etat de l'art

Pour comprendre l'inpainting des images, il est très utile d'étudier les réponses qui ont déjà été apportées. Des problèmes comme la restauration d'image ou les trucages simples comme la suppression d'éléments dans l'image ont ouvert la voie à une recherche efficace sur le sujet. La première tâche à effectuer était donc de produire un état de l'art complet sur les techniques d'inpainting des images. Cette état de l'art ne saurait être complet sans une mise au point sur les programmes et applications déjà existantes. En effet de nombreux chercheurs se sont déjà intéressés au problème de l'inpainting image et les programmes qui en résultaient devaient être examinés avec grand soin.

1.1.1 Etat de l'art théorique

Dans cette partie nous nous intéresserons tout particulièrement aux algorithmes [1](Criminisi, Perez, Toyama, 2004) et [3](Perez, Gangnet, Blake, 2004)qui furent testés et nous ferons un point sur les nouvelles techniques en devenir comme[6](Müller, Zeng, Wonka, Van Gool, 2007).

Historique

Dans les premières recherches sur l'inpainting des images, les recherches se sont concentrées sur des occlusions de petites tailles comme dans la restauration de vieilles images. Or les algorithmes ainsi développés s'adaptaient très mal à notre sujet, car nous travaillons sur des occlusions qui sont généralement assez grandes (comme des arbres) ou allongées (comme des lampadaires).
Ces méthodes sont principalement basées sur le lissage anisotrope ou des méthodes de détection et de propagation de T-jonctions, qui sont les intersections de deux structures (comme un coin de porte).

On retrouve ces algorithmes dans [7], or l'application de ce type d'algorithme sur les façades n'a donné aucun bons résultats, surtout au niveau des structures larges sur lesquelles elles ont tendances à supprimer les structures.
La deuxième génération d'algorithme travaillent quand à eux sur la synthèse de texture, comme dans les travaux de [2] que nous allons détailler ci-dessous.
La troisième génération a amélioré les systèmes précédent en se basant sur l'image pour remplir les occlusions. Cette méthode se sert de "patch", comme base de comparaison et ne remplit plus uniquement des pixels. De plus beaucoup d'algorithme ont rajouté des critères pour la sauvegarde de la géométrie tout en propageant la texture.
Ces méthodes sont bien évidemment beaucoup plus adaptées à notre problème. Nous allons ici en détailler trois d'entre elles.

Synthèse de Texture

Cette section fait surtout référence à l'article de Alexi Efros et de Thomas Leung [2] sur la propagations de texture à partir de textures existantes dans l'image. Cet algorithme propage les textures pixel par pixel et prend comme hypothèse que la modélisation de texture se fait comme un champ de Markov aléatoire.
Pour décrire l'algorithme, nous allons en expliquer le principe sur un pixel :
Soit I l'image à synthétiser à partir d'un exemple $I_{smp} \subset I_{real}$ (I_{real} étant l'image réelle de texture infini) et soit $p \in I$.
Soit de plus $w(p)$ une patch carré de grosseur w et $d(w_1, w_2)$ la distance entre deux patchs.
Pour synthétiser les valeurs de p, on construit une approximation de la distribution de probabilité conditionnelle :

$$P(p|w(p))$$

En se basant sur les champs de Markov on assume p indépendant avec $I/w(p)$.
On définit ainsi :

$$\Omega(p) = w' \in I_{real} : d(w', w(p)) = 0$$

Contenant toutes les occurrences de $w(p)$.
Le problème est que l'on a I_{sup} et non I_{real} on trouve alors :

$$\Omega'(p) \simeq \Omega(p)$$

On cherche alors la meilleure distance pour appliquer une méthode de plus proche voisin :

$$w'_{best} = argmin_w d_{perc}(w(p), w) \subset I_{smp}$$

Une fois celle-ci trouvée pour tous les patchs images w' tel que : $d_{perc}(w_best', w') < \epsilon$ (avec ϵ étant un seuil) inclus dans Ω', on peut propager le pixel. On choisi souvent d_{perc} comme étant une SSD normalisé (Sum of Square Differences metric) d_{SSD}.
Ainsi la valeur du pixel centrale de Ω' est la valeur donnée à p.
Nous pouvons voir sur des images de structures que cette algorithme marche bien comme montré ci-dessous :

Cette algorithme ne prend qu'un pixel a la fois et ne s'intéresse qu'à la texture, or nous avons besoin de prendre ne compte les structures dans un contexte de façade de bâtiment. Tout de même, cette algorithme fut testé même si les résultats trouvées n'ont pas permis de se contenter d'une simple synthèse de texture. Heureusement d'autres algorithme existent et marchent tout à fait correctement.
Toute fois, cet algorithme devait être présenté car il est la base de nombreux travaux sur la synthèse de texture, notamment dans l'inpainting.

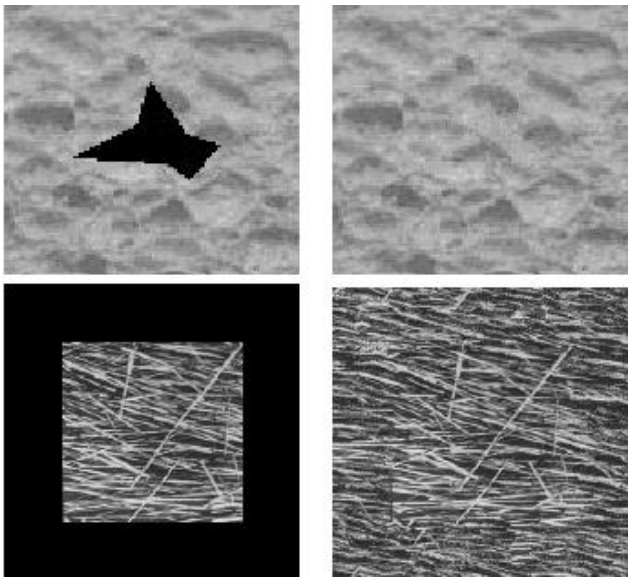
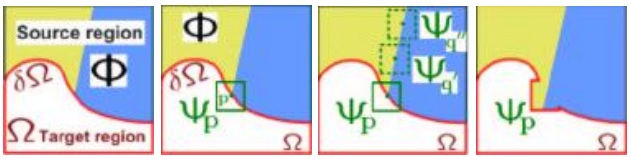


FIG. 1.1 – Résultat de l'algorithme de synthèse de texture ,Figures prisent dans l'article [2]

Tuilage à partir de morceaux de l'image (*Exemplar-based region tiling)*

Méthode de Criminisi-Perez-Toyama :

Cette algorithme est une avancée importante dans l'inpainting image. Il semble être tout à fait adapté à notre problème. Cet algorithme est appelé "Region filing and object remouval by Examplar-based Image Inpainting" ou en français "Propagation de Structure par Synthèse de Texture". En simplifiant , cette algorithme cherche à préserver les structure en propageant les texture le long des structures. Pour mieux comprendre on peut montrer ce schéma :



(a) situation initiale (b) choix du premier patch à remplir (c) selection des patchs tests (d) remplacement du patch

FIG. 1.2 – Étapes de remplissage du patch, ces schémas sont tirés de l'article [1]

Cette algorithme met en lumière deux points critiques dans les approches d'inpainting et pourquoi ces étapes ne doivent pas être négligées :

1. L'ordre de remplissage des patchs est critique, un mauvais ordre peut tout à fait propager une erreur mais surtout produire des structures géométriques imaginaires.
2. Le type de propagation est tout aussi critique, la méthode dite de la "pelure d'oignon" montre très vite sa limite et à tendance "tuer" les structures ou alors à en créer.

La conclusion de l'algorithme est qu'il faut d'abord remplir les endroits où se trouvent les structures et sur les zones de textures homogène appliquer la technique "pelure d'oignon".

Pour une meilleur compréhension de l'algorithme, nous allons le décrire point par point :

Soit l'image I, l'occlusion Ω, et $\Phi = I - \Omega$. Soit $\delta\Omega$ la frontière entre Φ et Ω.
$\forall$ pixel $p \in \delta\Omega$, on se donne :

1. une valeur (la valeur RGB du pixel).
2. une valeur de confiance.
3. une valeur temporaire de priorité.

1)Définir une priorité :

Soit Ψ_p le patch à remplir :

$$P(p) = C(p) * D(p) \tag{1.1}$$

Avec

1. $P(p)$ la priorité de p
2. $C(p)$ la confiance de p avec une initialisation à 1 $\forall p \in \Phi$ et à 0 $\forall p \in \Omega$

$$C(p) = \frac{\sum_{\tilde{p} \in (\Psi_p \cap \Phi)} C(\tilde{p})}{\mathcal{A}(\Psi_p)} \tag{1.2}$$

Avec $\mathcal{A}(\Psi_p)$ = l'aire de Ψ_p

3. $D(p)$ la donnée inertielle de p

$$D(p) = \frac{\left|\nabla I_p^{\perp}.np\right|}{\alpha} \tag{1.3}$$

Avec
- α un facteur de normalisation = 255 sur une image en niveau de gris
- $\nabla I_p^{\perp}$ est l'isophote, il est calculé comme la valeur maximum du gradient image dans la fenêtre $(\Psi_p \cap I)$
- np est estimé comme le vecteur unitaire orthogonale à la surface $\delta\Omega$

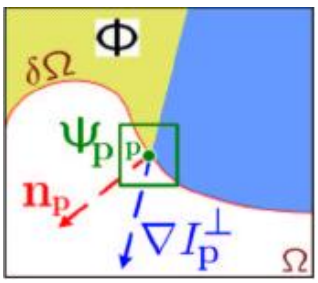


FIG. 1.3 – Schéma du calcul de valeur inertielle, cette figure est tiré de l'article [1]

2)Trouver le meilleur patch pour Ψ_p :

On cherche :

$$\Psi_q = \min_{p \in \Psi_p \cap \Phi} d(\Psi_p, \Psi_{\tilde{q}}) \tag{1.4}$$

la distance d choisie est le plus souvent SSD (Sum of Square Differences).
Une fois Ψ_q trouvé il ne reste plus qu'à remplir les pixels $p \in \Psi_p \cap \Omega$ avec ceux de Ψ_q

3)Mise à jour de la confiance

$$C(p) = C(\hat{p}) \qquad , \forall p \in \Psi_{\hat{p}} \cap \Omega \tag{1.5}$$

En conclusion on peut dire que cette méthode montre de bons résultats dans les exemples proposés car ceux-ci sont sous-résolus et alors cette technique marche parfaitement. Dans le cas d'image de très bonne qualité comme nos images de façades, le résultat fut décevant et cette algorithme ne fut pas utilisée. Mais il pose les bases de réflexions nécessaires et de questions sous-jascente à l'inpainting des images.

Méthode de Perez-Gangnet-Blake :

Finalement, nous présenterons, l'algorithme "Patchwork" Développé en 2004 par Perez, Gangnet et Blake et qui fut celui choisi pour ce projet. Cette algorithme, même s'il utilise une autre approche, est basé sur le système de remplissage de patch autour du masque d'occlusion.
Plusieurs différences existent, en effet cette algorithme se base sur un choix de patch uniquement déterminer par le nombre de voisins appartenant à l'image. Contrairement à précédemment où un facteur d'inertie permettait de donner plus de poids aux patchs sur une T-jonction. De plus ici on se donne un espace de recherche appelé dictionnaire contrairement à précédemment où la recherche de patch possibles s'effectuait à chaque patch traité.

Pour mieux appréhender cette algorithme nous allons le décrire de manière simple :

1. construction du dictionnaire des patchs $\subset I - \Omega$ qui seront les exemples
2. on construit les couches de remplissage dans Ω
3. pour chaque patch des couches on cherche le meilleur candidat dans le dictionnaire
4. on remplit la partie du patch $\in \Omega$ avec les valeurs des pixels du patch candidat

Nous allons maintenant expliquer cette algorithme tel qu'il l'est dans l'article des auteurs cités plus hauts.
$\forall$ pixel x est associé un patch p et un voisinage N, x étant toujours le point en haut à gauche du patch. On définit ainsi :

$$x + P = x + u, c \in P$$

$$x + N = x + u, u \in N$$

Soit D un ensemble de dictionnaire. $D = d_i$ est construit en considérant n patchs $P_i = x_i + P$ au point x_i avec $i \in [1, n]$

Un dictionnaire est en fait un vecteur

$$d_i = [n_i, p_i, x_i, i_i] \tag{1.6}$$

Où
– n_i = le vecteur des valeurs images des pixels $\in N$.
– p_i = le vecteur des valeurs de l'image auxiliaire dans les patchs.
– i_i = le vecteur d'indices de dictionnaires correspondant aux patchs du voisinage.

Il existe 8 possibilités de voisinage, chacun étant associés à une permutation dans le set T. Dans le cas d'un patch 2x2 et en 8-connexité $T = \{-2, 0, 2\}^2$

On peut ainsi explicité l'algorithme :

- Build sample dictionary $\mathcal{D} \doteq \{\mathbf{d}_i^*\}_{i=1\ldots n^*}$.
- Initialize completion mask $M = \{M(\mathbf{x}), \mathbf{x} \in \Omega\}$.
- While $M_0 \doteq \{\mathbf{x} \in \Omega : M(\mathbf{x}) = 0\} \neq \emptyset$ do:
  (a) Pick location j s.t. $P_j \cap M_0 \neq \emptyset$ and $N_j \not\subset M_0$.
  (b) Assemble $\mathbf{d}_j$ and $\mathbf{m}_j$.
  (c) Search: $\alpha(j) \leftarrow \mathtt{search}(\mathbf{d}_j, \mathbf{m}_j; \mathcal{D})$.
  (d) Copy: $J(P_j) \leftarrow I(P^*_{\alpha(j)})$, $M(P_j) \leftarrow 1$.

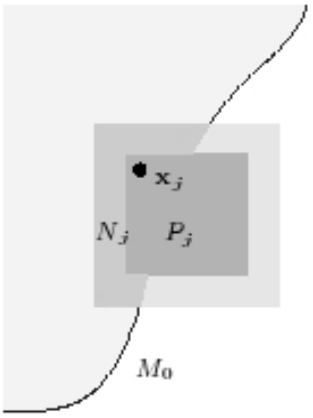


FIG. 1.4 – algorithme “Patchwork” tiré de [3]

La méthode “Patchwork” prend en entrée :

– p = grosseur du patch
– q = grosseur du voisinage
– l = la distance maximum du patch à sa destination
– J = image originale occluse supportant Ω
– Ω = support de J
– $D \subset \Omega$ = occlusion à remplir ($D \neq \varnothing$)
– $\Gamma \subset \Omega - D$ = source des patchs (partie non trouée de l’image)

Voici une figure explicative des différents paramètres cités ci-avant :

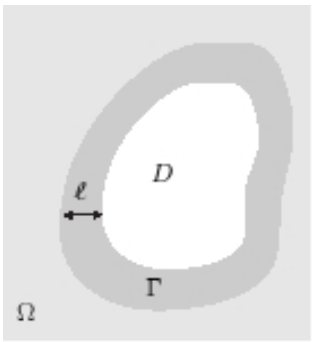


FIG. 1.5 – paramètres sur une image occlue tiré de [3]

⋆ Si $\Gamma = \varnothing, \Gamma \leftarrow \{x \in \Omega - D : d(x, D) \leqslant l\}$

⋆ $\forall x$ tel que $x + N \subset \Gamma$, on associe une entrée du dictionnaire

⋆ On initialise le masque M tel que :
$M(x) = 0$ ssi $x \in D$

$\star\ \Gamma : \Gamma \subset \sum_j P_j$, P_j ayant pour point top-left x_j

$\star$ On crée les courbes L^i avec $L^0 = \varnothing$
et $L^{k+1} = \{j \in \{1, \cdots n\} - \sum_{u=1}^{k} L^u \quad / \quad i_j \cap L^k = \varnothing\}$ pour $k = 0 \cdots K-1$

$\star$ Pour chaque couche lexicographique et pour un index de lieu j :

– On assemble d_j et m_j
– On cherche $\alpha(j) \leftarrow \arg\min_{i=1\cdots n} \|n_j - n_i\|_{1,m_j}$
– On copie $I(P_{\alpha(j)}) \rightarrow J(P_j), 1 \rightarrow M(P_j)$

Finalement de tous ces algorithmes, ce fut celui-ci qui fut utilisé. Les raisons de ce choix sont justifiées par les résultats, en effet il existe une multitude de programme sur Internet proposant de faire de l'inpainting, en me plongeant dans certains je me suis aperçu qu'ils implémentaient certains algorithmes vu précédemment. J'ai trouvé pour chacun un équivalent logiciel que j'ai pu tester sur un set d'images avec un set d'occlusions.

Avant de finaliser cette partie il faut de toute les manières faire un point sur les techniques les plus récentes en matière d'inpainting ou de reconstruction de façade. L'article que nous allons décrire ne parle pas "explicitement" d'inpainting mais la manière de reconstruire des façades est toute à fait pertinente et applicable à de l'inpainting. De plus cette méthode serait tout à fait nouvelle.
Ainsi nous allons parler de l'article de P.Müller, G.Zeng, P.Wonka, L.Van Gool [6] :
Le but de l'algorithme est : à parti d'une image de façade rectifiée on puisse en extraire les structures comme les étages, le nombres de fenêtres ... , structures qui ont tout à fait un sens sémantique fort.[1]

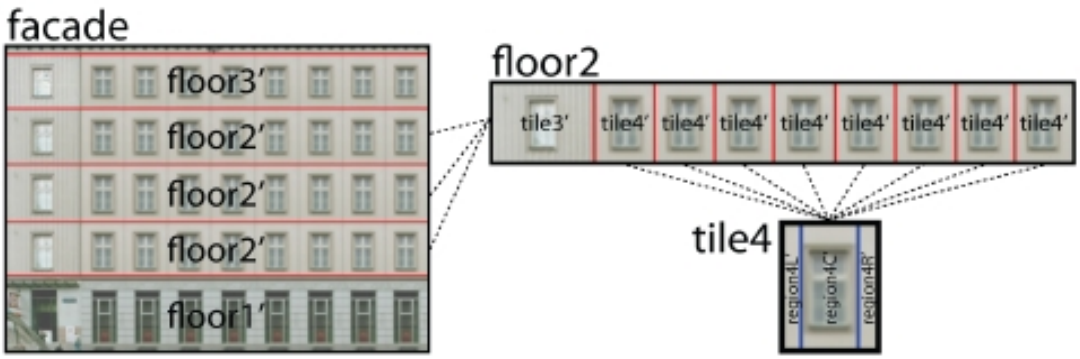


FIG. 1.6 – Résultat de la détection de structure sur une façade rectifiée[6]

Nous voyons ici le principe même de l'algorithme, que l'on peut vulgariser en quelques étapes :
– détection de structure sur les façades de plus en plus fines :
  – les étages
  – les "tiles" ou tuiles qui sont des blocs comprenant une fenêtre et des bouts de mur et qui se répètent
  – raffinement des tuiles en subdivisant en plus petites structures et en vérifiant quels tuiles sont similaires.
– élément de reconnaissance, en matchant ensembles des petites structures avec la bibliothèque d'architecture
– récupération de l'objet 3D

[1] les figures de façades ci-après sont toutes extraites de l'article [6]

– Édition d'une grammaire d'arêtes sur les façades

Si on vulgarise un peu moins on peu résumer l'algorithme comme une suite de 3 étapes que nous tenterons de décrire par la suite :

1. Détection de régions similaires dans l'image en utilisant les Informations Mutuelles (MI pour mutual information)
2. Création d'une donnée structure appelée : Façades Irréductibles (IF pour irréductible façade) qui nous permet d'encoder les informations sur la symétrie qui gouverne les étages et les Tuiles.
3. Analyse des IF pour trouver la subdivision optimal des Tuiles.

La clef de l'algorithme se trouve dans la détection des symétries translationnelles dans l'étape 2 avant de calculer les lignes de séparations dans l'étape 3.

Information Mutuelle (MI)

Définition 1.1.1.1. *: C'est la quantité qui mesure la dépendance mutuelle de deux variables quantitatives par la distance de Kullback-Leibler entre 2 distributions* $P(A = a, B = b)$ *et le produit des distributions marginales* $P(A = a)$ *et* $P(B = b)$*.*

Alors

$$MI(A, B) = \sum_{a,b} P(a, b) \log \frac{P(a, b)}{P(a).P(b)} \tag{1.7}$$

Avec A et B deux variables aléatoires.
On cherche à calculer la similarité de deux régions $\mathcal{R}_1$ et $\mathcal{R}_2$, pour cela il suffit d'appliquer la définition ci-dessus en adaptant avec l'histogramme joint et marginale des deux régions.

La Détection de Symétrie

On se sert de la MI pour trouver les étages et les tuiles similaires. En vertical, on attend une symétrie translationnelle des étages alors qu'à l'horizontale les étages sont redondant de tuiles.
⋆ Symétrie verticale
Soit $\mathcal{R}_{y,h}$ la région rectangulaire avec le point lower-left à $(0, y)$ et le top-right à $(widthing, y + h)$
On cherche les similarités entre $\mathcal{R}_{y_1,h}$ et $\mathcal{R}_{y_2,h}$ pour des valeurs arbitraires de y_1 et y_2 et h.
On peut simplifier le problème en ne prenant que des région adjacentes $\mathcal{R}_{y,h}$ et $\mathcal{R}_{y-h,h}$.
Or la similarité entre deux régions adjacentes est :

$$S(y, h) = MI(I(\mathcal{R}_y, h), I(\mathcal{R}_y - h, h)) \tag{1.8}$$

On fait un calcul exhaustif pour les positions de y avec un certain nombre de h, avec $3m < h < 5.5m$ pour tous les axes.

$$S_{max} = \max_h S(y, h)$$

Pour chaque lignes horizontales et

$$h_{max} = \arg\max_h S(y, h)$$

Les maximums correspondants indiquent la meilleure valeur de symétrie comme on peut le voir dans la figure 1.7.

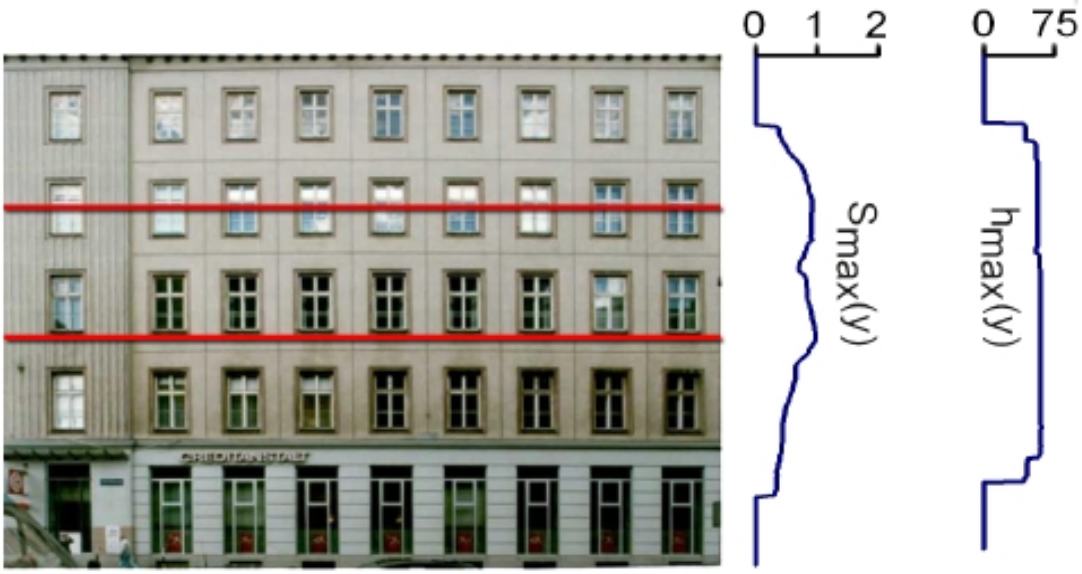


FIG. 1.7 – exemple de choix d'axe de symétrie

Les Façades Irréductibles (IF)

Définition 1.1.1.2. *Les façades irréductibles sont les miniatures contenant tout l'information de l'image*

En vulgarisant, c'est le plus petit morceau de façade qui ne peut être subdivisé. Pour calculer les façades irréductibles :

1. $IF(x, y)$ est initialisé pour être égale à l'image de façade
2. Itérativement on choisi la position : $y = \arg\max(S_{max}(y))$
3. On réduit l'image en supprimant la partie $\mathcal{R}_y - h_{max}(y)$
4. La liste des pixels au dessus est ajoutée à celle du dessous
5. On enlève de la même manière les symétries horizontales
6. L'algorithme se termine quand aucune symétrie supplémentaire ne peut être trouvé, c'est à dire qu'aucune valeur de $S_{max}(y)$ n'est supérieur à $0.75 * \tau_{max}$ avec τ_{max} comme meilleur score de similarité.

Subdivision de Structures

Nous recherchons à ce niveau les lignes verticales et horizontales :

$$ver(x, y) = \max\left\{\left(\frac{\partial I}{\partial x}\right)^2 - \alpha|\nabla I|^2, 0\right\}$$

En simplifiant on obtient :

$$ver(x, y) = \max\left\{(1-\alpha)\left(\frac{\partial I}{\partial x}\right)^2 - \alpha\left(\frac{\partial I}{\partial y}\right)^2, 0\right\} \tag{1.9}$$

$$horiz(x, y) = \max\left\{(1-\alpha)\left(\frac{\partial I}{\partial y}\right)^2 - \alpha\left(\frac{\partial I}{\partial x}\right)^2, 0\right\} \tag{1.10}$$

Avec $\alpha = 0.9$ (par défaut). La préférence pour les séparations horizontales en y ou verticales en x se fait séparément des valeurs des deux fonctions d'intérêt :

$$Ver(y) = (\sum_x ver(x,y)) * g_\sigma(y) - \beta(\sum_x horiz(x,y)) * g_\sigma(y) \tag{1.11}$$

$$Hor(x) = (\sum_y horiz(x,y)) * g_\sigma(x) - \beta(\sum_y ver(x,y)) * g_\sigma(x) \tag{1.12}$$

Où
$g_\sigma(.) = \frac{1}{2\pi\sigma^2} e^{-\frac{|.|^2}{2\sigma^2}}$
$\beta = 0.1$ (par défaut)
$\sigma = 1m$ (par défaut)
Finalement, une recherche exhaustive pour trouver les positions potentielless optimales $Y_i \subset y_i$ avec la contrainte d'une connaissance à priori sur la hauteur des étages.

$$Y_i = \arg\min_{\{\hat{y_i}\}} \frac{\sum_i Ver(\hat{y_i})}{\|\{\hat{y_i}\}\|} \tag{1.13}$$

Avec $\{\hat{y_i}\} \subset \{y_i\}$ et $3 < \nabla\hat{y_i} < 5.5$ et $\nabla\hat{y_i} = y_{i+1} - \hat{y_i}$

$$X_i = \arg\min_{\{\hat{x_i}\}} \frac{\sum_i Hor(\hat{x_i})}{\|\{\hat{x_i}\}\|} \tag{1.14}$$

Avec $\{\hat{x_i}\} \subset \{x_i\}$ et $0.5 < \nabla\hat{x_i} < 9$ et $\nabla\hat{x_i} = x_{i+1} - \hat{x_i}$

Subdivision en Tuiles

On a récupéré les tuiles, il faut maintenant re-subdiviser les tuiles en IF. On créer pour cela une hiérarchie d'éléments dont voici l'algorithme :

- Initialisation de toutes les tuiles
- Tant que la région traité n'est pas une feuille ET que la région de gauche n'est pas subdivisée :
  - Trouve le meilleur candidat de séparation pour chaque région
  - Synchronisation de la séparation des tuiles du même groupe
  - Synchronisation globale de tous les candidats de séparation
  - Subdivision des régions en nouvelles régions plus petites
  - Marquage des régions non subdivisable en feuilles
  - marquage des petites régions comme feuilles

Conclusion

On imagine très bien les possibilités que nous offrent cet algorithme en terme de tuiles semblables. Imaginons le problème de l'inpainting de façade à l'aide des outils proposé dans cette article. Alors une occlusion qui se ferait sur une partie de tuiles pourrait tout a fait être remplie en cherchant dans la base une tuile semblable et en la remplaçant tout simplement. Pour choisir une métaphore tout à fait approprié, lorsque dans un toit on a un trou, on remplace le trou avec des tuiles ressemblant à celles voisines. De plus, le rêve de tout couvreur serait de pouvoir copier les tuiles d'à coté pour les remplacer à l'endroit du trou.
Cette algorithme, même s'il n'est pas en relation "direct" avec l'inpainting, apporte un horizon nouveau pour l'inpainting image.

1.1.2 Etat de l'art logiciel

De nombreux algorithmes ont été développés sous forme de programme trouvable sur Internet. Nous nous proposerons d'en tester quelque uns. Nous allons séparer ce chapitre en deux parties distinctes. Dans la première, nous expliquerons le banc de tests choisi et les raisons de ces choix. Dans la seconde, nous donnerons les résultats de ces tests sur les programmes que nous avons choisi.

Le banc de test

1) Le choix des images
La première étape de la construction du banc de tests fut le choix des images. J'ai choisi alors des images de la façade de l'école des mines de Paris prise par temps clair sans rectification. Ce type d'image offre l'opportunité de tester le programme sur une image avec des structures complexes comme des moulures et des textures répétitives comme la brique ou les fenêtres à une échelle différente.
Pour la fabrication des occlusions j'ai utilisé un programme C marchant sous la bibliothèque megawave. Ce programme permet de dessiner facilement, rapidement et à la main des occlusions. Bien sur dans le cadre du projet cette étape est censée avoir été effectuée auparavant et de manière automatique.
Les images retenues furent :

Nom	Représente	Résolution	Taille
facademines1.jpg	façade d'immeuble ancien avec peu de répétition de structure	haute	900 x 1200
facademines2.jpg	façade d'immeuble ancien avec beaucoup de répétition de structure	haute	900 x 700
facades1.jpg	façade de maison avec peu de répétition	moyenne	399 x 271
facades2.jpg	façade d'hôtel avec beaucoup de répétition	basse	590 x 583
facades3.jpg	façade de débit de boissons	basse	400 x 250
facades4.jpg	façade d'immeuble avec beaucoup de répétition	haute	370 x 170
facademuller.tif	façade d'immeuble moderne totalement répétitive	basse	1200 x 800

TAB. 1.1 – Définition des images test

Une fois cette échantillon choisi, il faut trouver les meilleurs occlusions possibles. Choisir un algorithme ne se fait pas facilement et là où un algorithme donne de très bons résultats, un changement d'images ou d'occlusions permettent de voir la permittivité des algorithmes.

2) Le choix des occlusions :

Ci-dessous les types d'occlusions choisis et pourquoi :

⋆ Une zone de texture unie, comme par exemple une zone de ciel.

⋆ Zone de texture, comme par exemple un morceau de mur en brique.

⋆ Zone de forte géométrie comme par exemple couper un morceau de tuyau ou alors un petit bout de croisé de fenêtre.

⋆ Zone à problème comme des coins de fenêtre ou des cheminées ou alors enlevé une grande partie d'une structure en ne laissant qu'un coin.

Le choix du programme :

Nom	Auteur	Support	Algorithme
Inpaint7	Sooraj Bhat	Matlab	Criminisi
ImageRestoration	collectif	Java,C++	Efros
Inpainting	Hamilton Chong	C++	Efros
inpaint	Zhang	C++	criminisi
Resynth	Paul Harrison	GIMP	Harrison
reproduce	Yann Gousseau	C, megawave	Perez

TAB. 1.2 – Récapitulatif des programmes gratuits trouvés sur Internet testés

Nom	plus	moins
Inpaint7	bonne pour géométrie	LENT
ImageRestoration	bien pour craquelure	mauvaise sinon
Inpainting	bien sur texture	mauvais pour la géométrie
inpaint	bon sur texture et un peu géo	mauvais sur les façade hr
Resynth	bon pour craquelure	mauvais pour le reste
reproduce	plutôt bon et rapide	tendance à détruire la géométrie

TAB. 1.3 – Plus/moins des programmes gratuits du net

Le choix fut porté sur “reproduce”, application développée au sein du laboratoire TSI et qui semblait beaucoup mieux marcher avec les images de façades hautes résolutions de l'école des mines de Paris.
Mais l'utilisation du programme m'a fait comprendre à quel point le choix des paramètres est décisif dans les résultats de l'algorithme.
Ainsi pour améliorer ce code nous avons rajouté plusieurs options qui améliorent sensiblement les résultats et qui seront définies dans le prochain chapitre.

1.2 Innovations

Dans cette partie, nous tenterons de démontrer l'efficacité de nos recherches sur le problème délicat de l'inpainting d'images de façades. Les innovations sont basées sur l'observation des images de façades qui nous a permis de mettre à jour des idées simples. celles-ci ont permis de mettre en place deux des algorithmes expliqués ci-après, la symétrie et la préférence horizontale. Même si l'observation est la base de toutes les innovations d'autres idées sont venus de l'expérience. Ces innovations sont le fruit de réflexions conjointes entre Michel Roux, mon maître de stage et ayant travaillé sur la reconstruction 3D de bâtiments, et moi-même. Le travail effectué sur les algorithmes a permis de mettre en évidence les points critiques d'un algorithme d'inpainting image :

1. l'ordre des patchs traités, dans certain cas ce problème peut complètement détériorer la géométrie des images et perdre en cohérence.

2. la zone de recherche optimum pour trouver le meilleur correspondant. ce problème peut, s'il n'est pas traité, ralentir sensiblement le programme même si alors on y gagne en fiabilité.

3. la mesure choisie pour trouver le bon correspondant.

1.2.1 Symétrie

Observation

Paris est une ville extraordinaire pour la multitude de ses architectures et pour l'omniprésence de l'architecte qui a révolutionné le Paris médiéval pour en faire la ville lumière, c'est à dire le Baron Haussmann. Celui-ci créa un certain nombre d'avenues et son style fût répété. Lorsqu'on travail sur la reconstruction de façades ainsi qu'à l'inpainting de celles-ci ,et particulièrement dans le cadre d'un projet comme Terranumerica, qui souhaite réaliser ces travaux à grande échelle, il est nécessaire de s'intéresser aux façades, c'est à dire à la matière première de ces algorithmes. Or le Baron Haussmann avait un désir de ligne droite et de symétrie. Les immeubles modernes sont quant à eux assez simples et possèdent peu de structures complexes et possèdent traditionnellement de nombreuses symétries. Voila pourquoi nous nous intéresserons aux bâtiments plus complexes(avec les moulures, balcons et sculptures qui composent les façades), comme ceux du Baron Haussmann.
Par une simple observation de ces façades on remarque que les fenêtres sont symétriques ou que chaque façade Haussmanniennes comporte un axe de symétrie dans l'axe de la porte d'entrée. Cette simple remarque nous a fait poser la question suivante : Imaginons que la moitié de la façade soit occultée, alors il suffirait de faire une symétrie de l'autre partie et on retrouverait le bâtiment original.

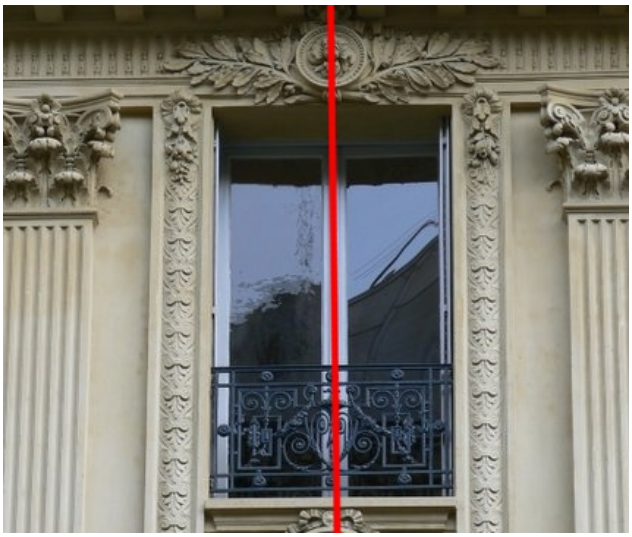


FIG. 1.8 – Axe de symétrie dans une fenêtre de style Haussmannien

Mise en place

La première étape fût de créer l'image symétrique, ce qui n'est pas difficile. Une fois que l'on a l'image symétrique, on est en droit de se demander comment les patchs vont aller chercher l'information contenue dans l'image symétrique. Pour cela il faut se reporter à la figure 1.5, on remarque alors que l'espace du dictionnaire (compris entre I et D) ne se fait que sur l'image originale. Il faut donc un dictionnaire se basant sur l'image originale et sur l'image symétrisée en se servant par exemple de deux dictionnaires intermédiaires. Il suffit de prendre le dictionnaire de patchs possibles, de symétriser leurs coordonnées et de choisir comme source des indices l'image symétrique.

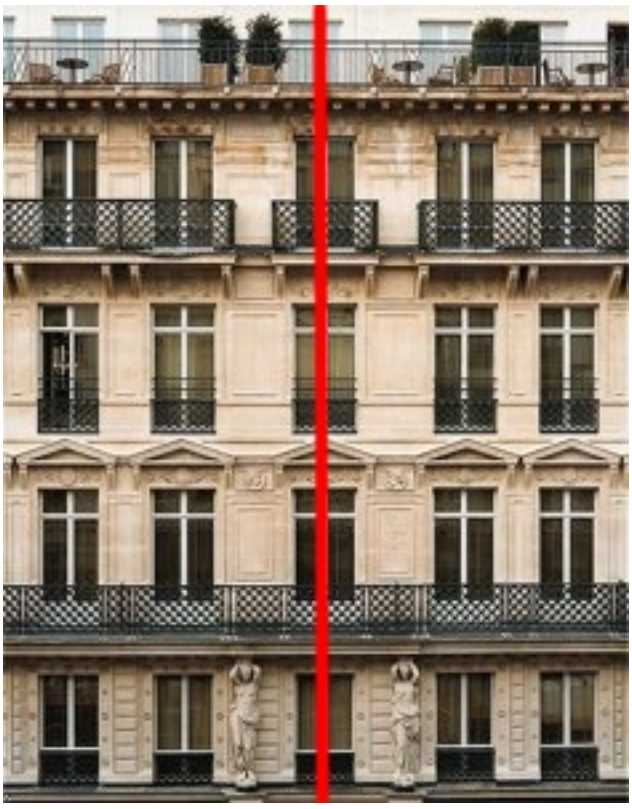


FIG. 1.9 – Axe de symétrie dans une façade de style Haussmannien

1.2.2 Préférence horizontale

observations

Sur n'importe quel bâtiment d'habitat que vous regarderez sur paris, vous remarquerez et surtout dans les bâtiments Haussmanniens que les lignes architecturales sont plutôt horizontales et quelques fois verticales, mais à part quelques exceptions jamais diagonales ou dans des directions exotiques.
Cette observation découle complètement de la précédente, en effet, si les axes de symétries sont plutôt verticaux, alors il est tout à fait légitime de se dire que cela implique une recherche privilégiée dans la direction horizontale et dans certain cas verticale. Cette recherche s'effectue à l'aide d'un dictionnaire différent ainsi qu'un parcours de celui-ci différent.
Dans l'algorithme classique, le dictionnaire de patchs possibles est choisi autour de l'occlusion. la distance autour de celle-ci est dépendante des paramètres d'entrée mais quoi qu'il arrive à tout patch de l'occlusion on testera tous les patchs voisins de l'occlusion et non pas du patch traité.

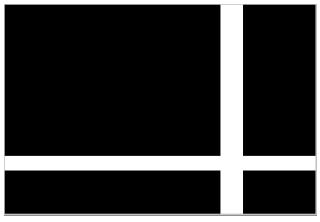
mise en place

Dans ce contexte, la forme du dictionnaire change car il devient rectangulaire autour du patch, c'est à dire que pour la création du dictionnaire de patchs possibles, au lieu de ne chercher qu'autour de l'occlusion, on cherche sur toute la longueur de l'image avec pour bord verticaux, la hauteur du trou. Mais en faisant ça on augmente de manière quadratique le temps de calcul car la taille du dictionnaire croit de même. Ainsi pour gagner du temps et parce que c'est plus logique, on ne cherche qu'à la hauteur du patch traité. Ne pas suivre ce principe reviendrait dans le cas d'un immeuble à essayer de remplir une fenêtre avec du ciel.

(a) masque originale

(b) espace de recherche originale

(c) espace de recherche total horizontale

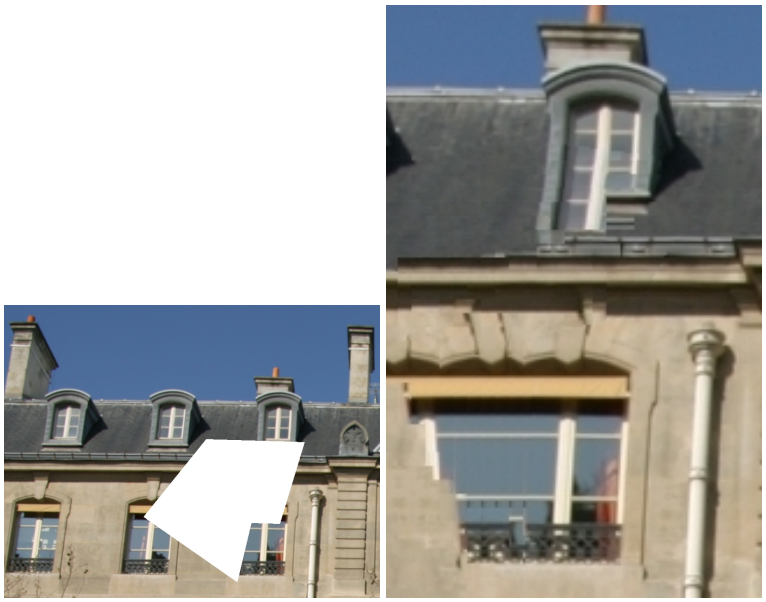


(d) espace de recherche total horizontale et verticale

1.2.3 Minimisation d'énergie par ICM

Observations

En étudiant les résultats obtenus grâce aux deux précédentes observations, on s'aperçoit qu'un problème subsiste. Il est inhérent à l'algorithme et ne peut être modifié d'aucune manière. C'est la propagation des erreurs le long des géométries. En effet sur un coin de fenêtre retrouver l'angle droit se révèle quasiment impossible. Dans la majorité des cas, la géométrie passe en premier et elle se propage plus rapidement que la simple texture. De plus privilégier la direction horizontale impose que les lignes horizontales seront traitées en priorité. Dans le cas d'angle, comme un angle de fenêtre, le résultat est de mauvaise qualité dans le sens où on ne retrouve pas la géométrie originale mais une géométrie complètement déformée.



(e) masque d'occlusion

(f) détail de problème de propagation de structure

FIG. 1.10 – Exemple de propagation de structure

Pour résoudre ce problème, beaucoup de questions se sont posées. l'idée pour résoudre ce problème est de se dire, une fois le premier traitement fait de revenir dessus et de rechoisir le meilleur candidat en fonction des voisins nouvellement choisis. Pour le choix du nouveau meilleur patch, une minimisation d'énergie par ICM fut choisi. L'idée est que si l'énergie est minimum avec un candidat, alors il est le meilleur. Ainsi en remplaçant un patch on influe sur toute une zone. Ainsi par itération successives sur tous les patchs ont approche une solution optimum. L'utilisation de l'ICM est liée au fait que l'algorithme converge rapidement et en temps limité vers un minimum qui peut être globale ou locale.

mise en place

Tout d'abord il faut définir une mesure d'énergie, celle-ci aurait put être le recouvrement si les patchs se recouvraient, or il ne le font pas. Ainsi l'idée fût de grandir les patchs et de considérer le recouvrement entre chaque gros patchs et ses voisins eux aussi grossis. Mais la mesure en elle-même est une mesure de corrélation. Or cette corrélation ne pouvait se faire au niveau de l'occlusion car celle-ci est censée être vide (on ne compare pas du blanc avec du blanc). Nous avons ainsi mis au point une mesure calculant la corrélation sur les parties se chevauchant mais appliquée aux correspondants. Pour mieux comprendre voir le schéma ci-dessous.

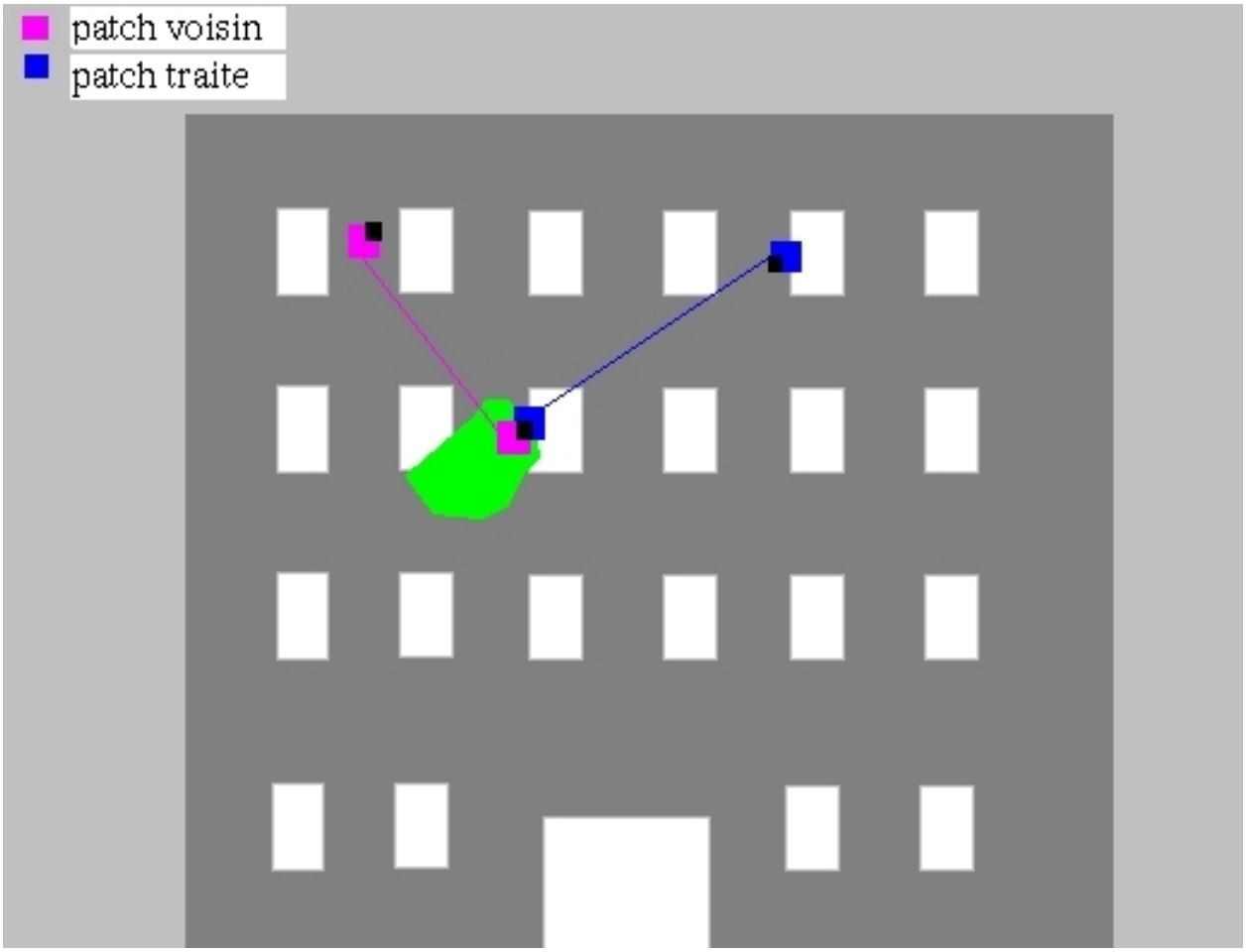


FIG. 1.11 – Exemple de mesure

Une fois cette mesure mis en place nous devons nous en servir pour calculer une première fois l'énergie. Ceci fait on calcul la mesure précédente sur tous les patchs pour un certain nombre de candidats sauvegardés. Pour une meilleure robustesse, il faudrait tester à chaque patchs tous ses candidats, mais pour des raisons de taille mémoire et de complexité, nous avons choisi de n'en garder qu'une dizaine.

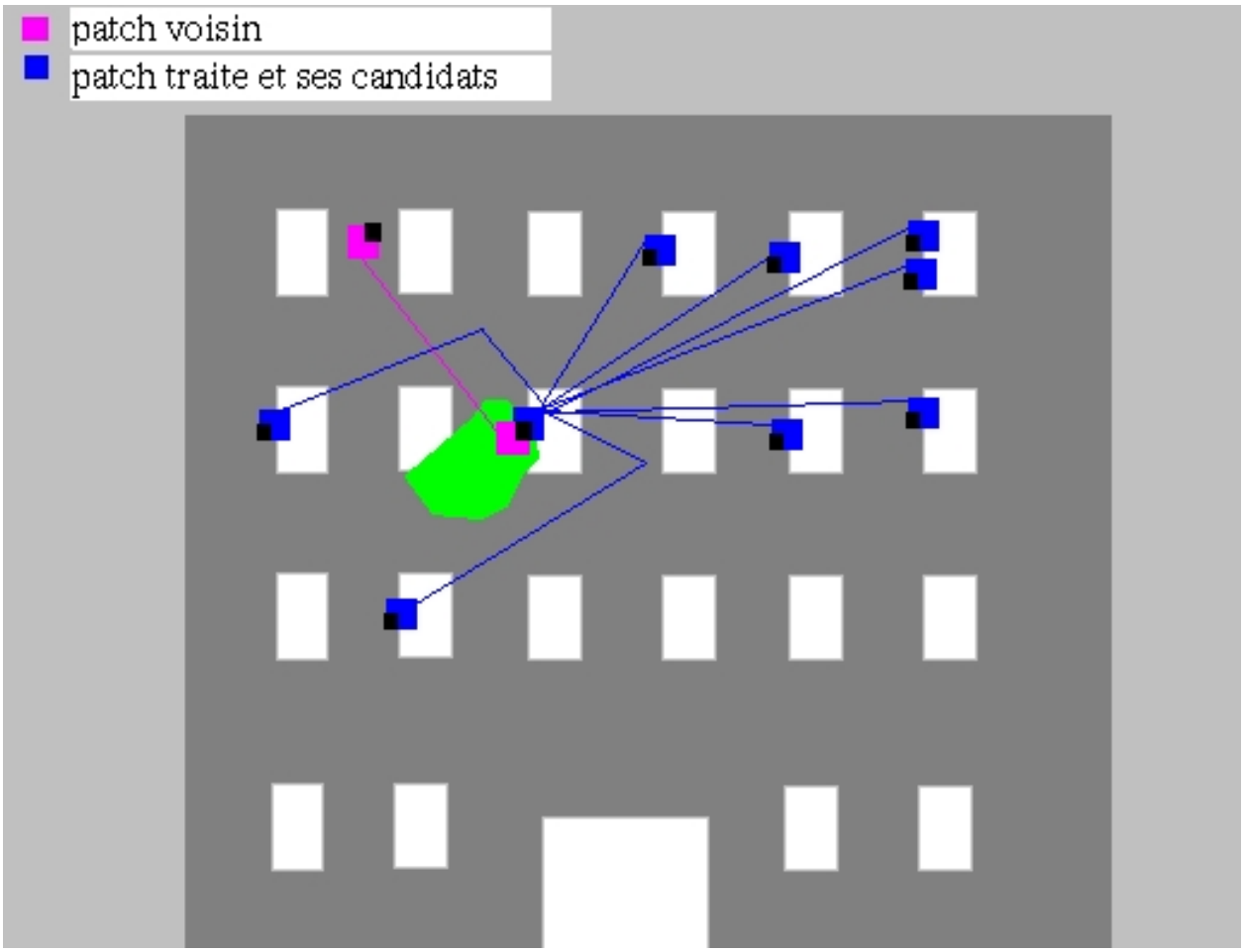
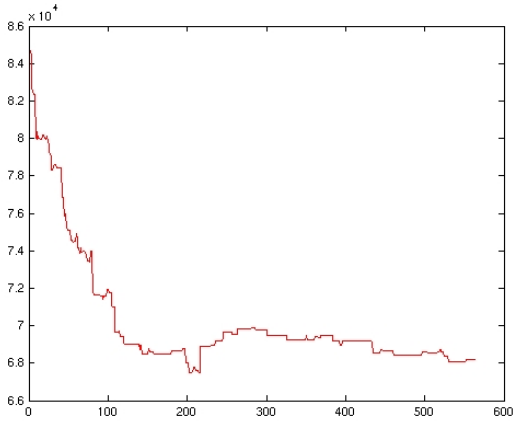
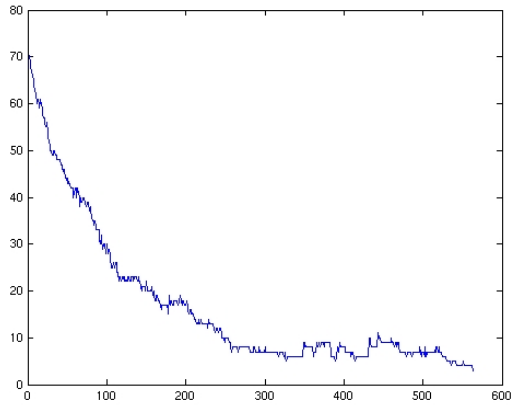


FIG. 1.12 – Exemple de mesure sur les candidats

Ce type de recherche ne se fait pas qu'avec un voisin mais avec tous les voisins qui chevauchent le patch traité. Ainsi une fois le meilleur candidat choisi, on remplace l'ancienne valeur du patch par sa nouvelle. On applique cette algorithme pour tous les patchs de l'occlusion. Une fois l'ensemble des patchs traités, on regarde si l'énergie décroît. L'algorithme s'arrête quand le nombre de changement entre chaque itération devient inférieur à 3 pourcents du nombre total de patchs. Cet algorithme ne change rien si les meilleurs patchs sont ceux choisis la première fois. On peut voir la progression de l'énergie ainsi que celle du nombre de changements sur un exemple avec de petits patchs et une grosse occlusion(cas qui donne beaucoup plus d'itérations qu'un cas simple et permet de mieux voir le phénomène. Normalement seulement 4 à 10 itérations sont nécessaires, ici il y en a plus de 500).
Le résultat quand à lui montre à quel point cet algorithme peut montrer de bonne choses, malheureusement sa tendance a pixeliser la désocclusion nous force trop souvent à ne pas s'en servir. Dans de nombreux cas, cette pixelisation s'il on peut dire nuit à la cohérence de l'image. De plus le temps de calcul peut parfois être long compte tenu du gain et qui peut s'avérer mauvais.



(a) descente d'énergie



(b) descente de changement

FIG. 1.13 – Résultat de l'ICM sur le même exemple que 1.10

1.2.4 Sous-échantillonnage

observation

Les images que nous devons traiter sont de grandes tailles et posent des problèmes quand au temps de calcul. De plus il est vrai que les lignes sont mieux comprises par l'algorithme lorsqu'elles sont plus fines et les structures géométriques sont mieux rendues. Les tests effectués ont donné de meilleurs résultats quand les images étaient plus petites. Il était donc légitime d'envisager de sous-échantillonner les images. On dégage deux avantages à cet algorithme :

1. le temps de calcul qui diminue, sous échantillonner revenant à travailler sur un plus petit nombre de données.
2. la fiabilité, le sous échantillonnage permet un affinement des contours.

Nous avons choisi de travailler parallèlement sur l'image sous-résolue et normale. Dans un premier temps un travail sur l'image sous-résolue tel qu'on a pu le voir dans les chapitres précédents sera effectué, puis une fois les correspondances gardées en mémoire il suffira de refaire le même travail sur l'image d'origine mais dont le dictionnaire sera restreint autour du meilleur correspondant de l'étape précédente. Évidemment tous les indices sont mis à échelle quand on passe de l'image sous-résolue à l'image d'origine.
Mais cette mise à échelle provoque une incertitude quand à la véritable position du correspondant dans l'image d'origine. C'est la raison pour laquelle nous effectuons un deuxième travail de corrélation appliqué à l'image d'origine, pour éviter des erreurs de décalage.

mise en place

La mise en place de la première phase fut assez simple, car il suffisait de reprendre le programme déjà écrit et de rajouter une fonction sous-échantillonnant l'image. J'ai décidé de séparer le programme en deux parties distinctes. La première fut mise en place de manière rapide et efficace. Mais la deuxième fut plus complexe en effet une fois le meilleur patch sur l'image sous-résolue, que l'on appellera par abus de langage imagette, obtenu il faut maintenant rapporter ces coordonnées sur l'image originale, que l'on appellera HR pour haute résolution, mais une imprécision existe qu'il faut soulever, ainsi on commence par faire une corrélation pour trouver le meilleur point possible comme centre du patch. Pour cela on se sert d'un dictionnaire mais beaucoup plus petit :

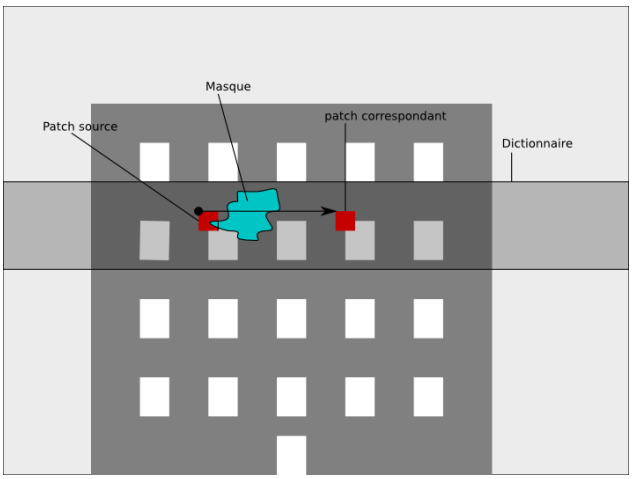


FIG. 1.14 – Exemple du choix d'un patch dans un dictionnaire sur l'imagette

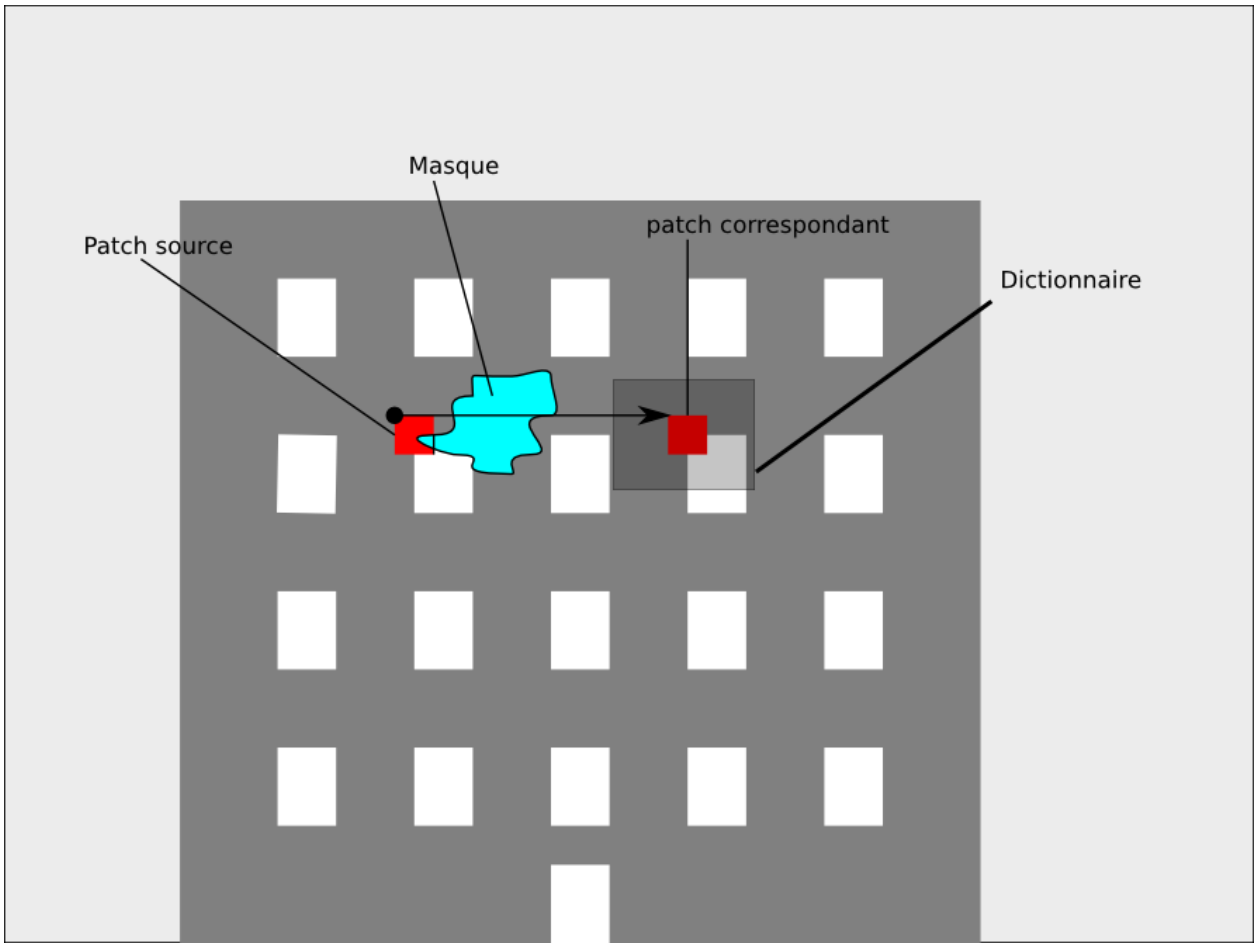


FIG. 1.15 – Même exemple mais sur l'image HR sur un dictionnaire réduit

Nous avions décidé de permettre un choix de taille pour la grosseur des patchs libres. Ainsi on pouvait tout à fait sélectionner une taille de patch différente pour l'imagette et pour l'image HR. Pour exemple si le facteur de sous-résolution est n alors si la taille choisie de patch pour l'imagette est de t alors la taille sur l'image HR $T \neq ou = n * t$. Mais à chaque cas possible, le traitement est différent, ainsi cette partie correspond en fait à trois algorithmes :

– Si $T < n * t$
– Si $T = n * t$
– Si $T > n * t$

FIG. 1.16 – Exemple avec une taille t sur l'imagette

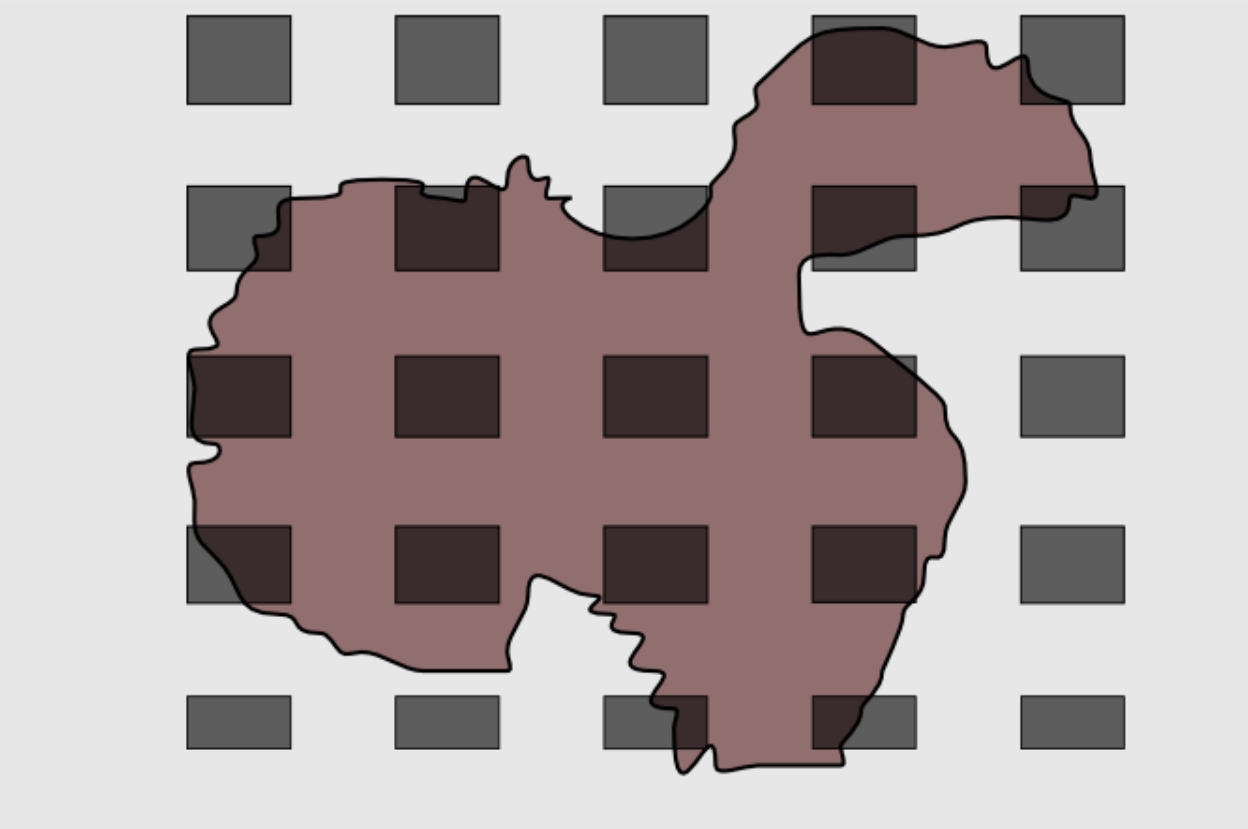


FIG. 1.17 – Exemple de problème si $T < n * t$ sur l'image HR

Pour palier à ces problèmes, nous avons décidé que sur la grande image on ferait la construction de patchs indépendamment de l'imagette. Pour les correspondants autres que ceux du point en haut à gauche, il suffit de déplacer le dictionnaire ou alors tout simplement de créer un dictionnaire de taille $n * t$:

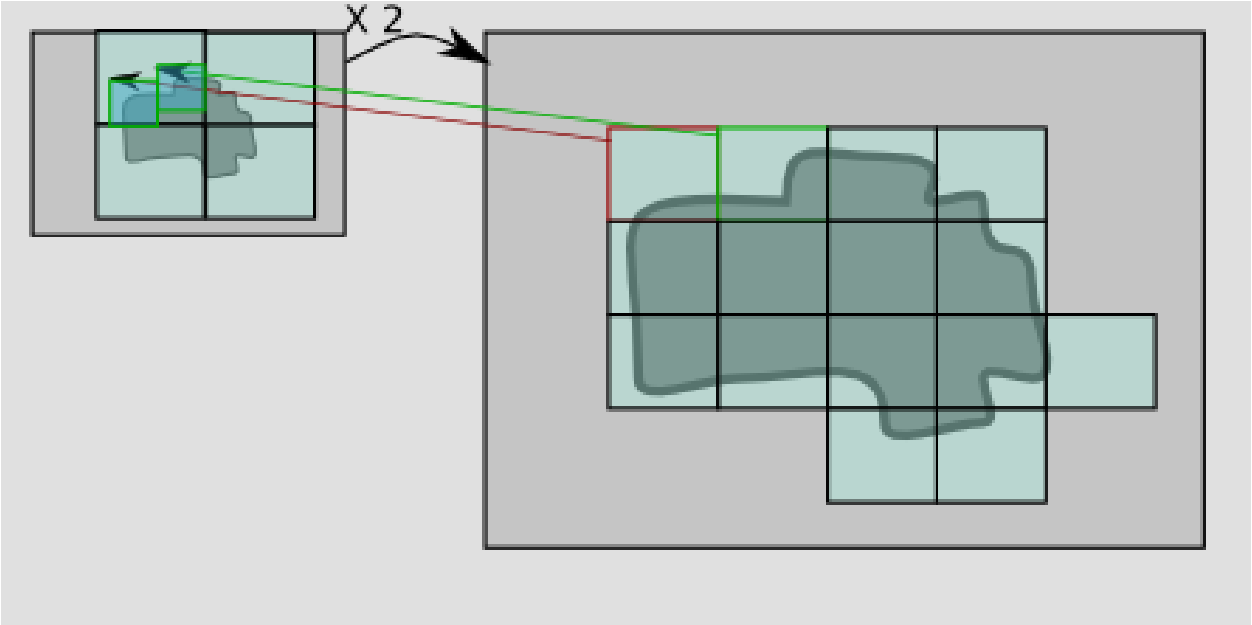


FIG. 1.18 – Exemple de construction de couches de patchs indépendantes et un exemple de mise en correspondance entre les deux

De la même manière si nous devons remplir des patchs plus grands que ceux de la petite image au paramètre de facteur de sous-échantillonnage près, alors il suffit de faire la même opération. Mais deux optiques existent dans la résolution de ce problème :

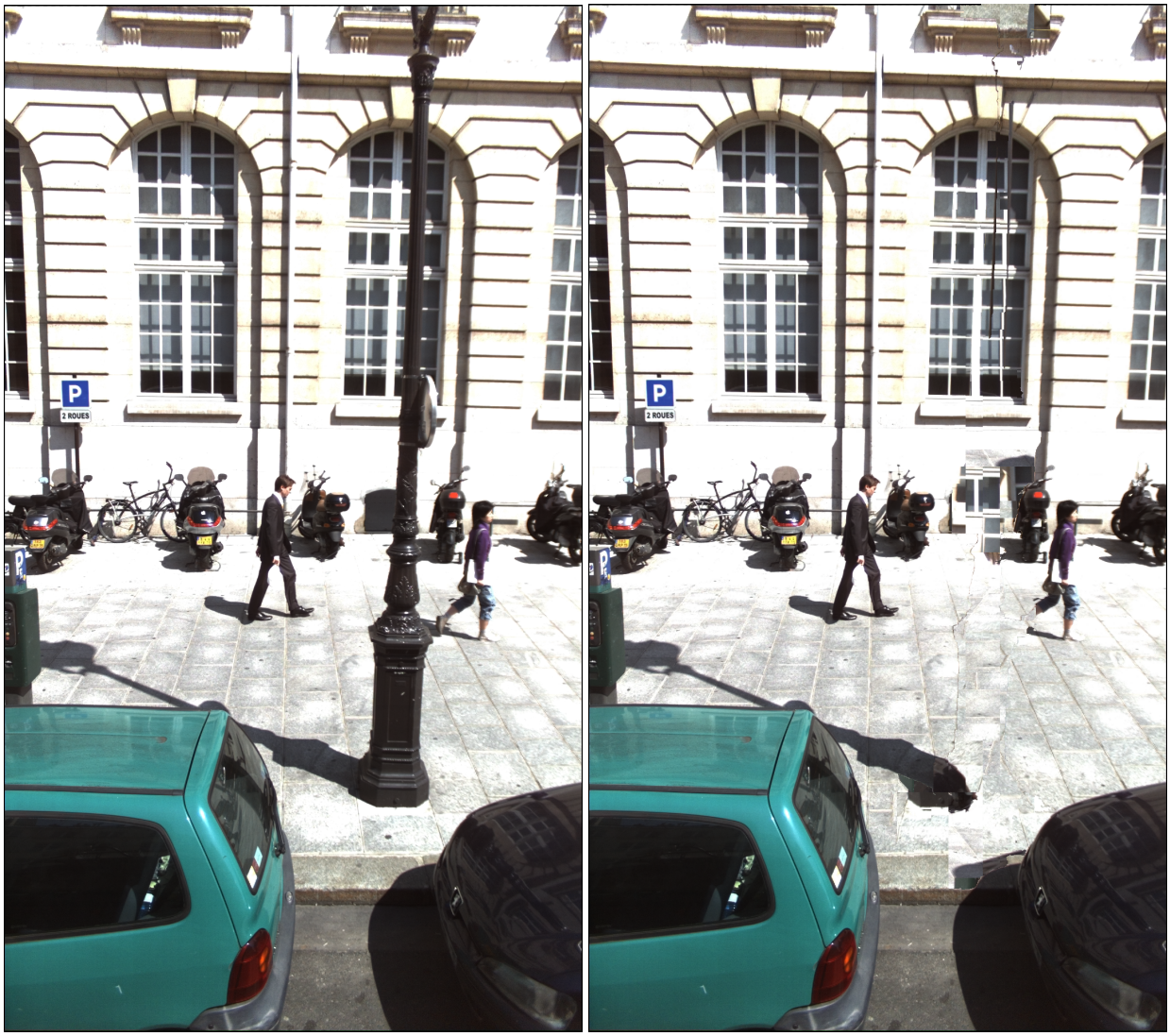
1. La première est de remplir tout se qui est possible avec le premier patch et ensuite de remplir l'occlusion restante sur le patchs à l'aide des autres correspondance
2. La deuxième est de ne conserver que la première correspondance et juste d'agrandir la zone de recherche.

Or c'est cette deuxième méthode que nous avons conservé car elle offre les meilleurs résultats, dans le premier cas, une disjointure pouvait apparaître.
Cette partie de sous-échantillonnage, malgré sa simplicité, nous a offert la meilleure progression par rapport aux travaux précédents. En effet nous avons put vérifier de manière tout a fait empirique que cette méthode donnait de bien meilleurs résultats dans tous les cas. De plus, lié au travail sur les imagettes, le temps de calcul est diminué ce qui est un avantage non négligeable.
Enfin je terminerais en montrant deux exemples de résultats qui marchent bien :

1. Une image avec un masque important mais une grande fréquence de géométrie
2. Une image avec un masque plus fin créer de manière automatique mais avec peu de fréquence.

(a) image masqué (b) image inpaintée

FIG. 1.19 – Résultat de l'inpainting sur une image de façade d'immeuble



(a) image masqué (b) image inpaintée

FIG. 1.20 – résultat sur une image de façade dont l'occlusion est calculé automatiquement

Chapitre 2

Inpainting 3D

Introduction

Dans cette partie nous allons aborder le problème délicat de l'inpainting 3D. L'état de l'art sur ce problème est pour ainsi dire inexistant tant ce type de technologie est novatrice, de plus rien dans ce que j'ai put lire ne m'a permis de l'appliquer au problème de façade qui nous occupe. Le seul état de l'art que je pourrais citer est un livre de géométrie dans l'espace. La plupart des algorithmes étudiés ne se servent que du nuage de points pour densifier ou estimer des paramètres, or nous avons à notre disposition l'image spectrale de la scène observée. La nouveauté réside donc dans l'utilisation conjointe de l'information spectrale et LIDAR pour résoudre ce problème d'inpainting 3D. L'idée première fut de se dire que si l'inpainting image donne de bons résultats, il serait intéressant alors d'utiliser ces méthodes pour l'inpainting 3D. Cela peut paraître simple mais difficile à mettre en oeuvre, ainsi le but final de cette inpainting 3D est de chercher à créer une image de profondeur sur laquelle nous tenterons après d'inpainter les parties manquantes correspondantes aux ombres LIDAR.

2.1 Projection 3D sur 2D

Pour comprendre comment peut-on projetter des points 3D dans une image on se référera à l'annexe, ré-expliquant les bases de l'optique et de photogrammétrie.
Pour permettre une projection, plusieurs choses sont necessaires comme évidemment le géoréférencement de toutes les données, c'est à dire que toute donnée doit pouvoir être placée dans un repère commun aux autres données. Sur le projet qui nous occupe, nous travaillons avec des images et données LIDAR prisent par l'IGN. Or ces données étaient de très bonnes qualitées et surtout, elles étaient fournies avec des programmes permettant l'exploitation simple et rapide des fichiers de configuration en XML fournis. Ainsi, la bibliothèque C++ fournie avec les données a permis de comprendre comment utiliser les données dans le cadre de projection simple sur l'image. Mais, cette bibliothèque tout en étant performante manque cruellement de documentations ce qui fut un frein au début de son exploitation. De même, l'exploration des codes sources ne m'a fournit aucun élément,aucun commentaire permettant de comprendre le but et l'utilisation des différentes fonctions. Ainsi un travail de compréhension de code fut fournit pour pouvoir utiliser ces fonctions. Finalement une fois le code compris, son utilisation s'avéra assez simple n'utilisant qu'un parcours simple du fichier caméra, ainsi que des coordonnées des points LIDAR. On a put ainsi projeter les points LIDAR sur l'image et ainsi mettre en lumière la répartition des points 3D sur l'images. Cela nous a permis de voir que le nuage n'était pas assez dense par rapport au nombre de pixel, en moyenne, on a seulement un point 3d tous les 12*12=144 pixels.

De plus cette étape nous a permis de créer automatiquement le masque d'occlusion pour l'image contenant le lampadaire. Michel Roux m'a fournit un nuage de points filtrés contenant uniquement les points lampadaires. Une fois ceux-ci projettés, il nous suffit de prendre l'image de segmentation et d'oc-

clure les zones contenant au moins un point lampadaire et qui ne sont pas trop éloignés d'au moins un point lampadaire. Le résultat à la figure 1.20 nous montre le résultat sur une image dont le masque a été créé automatiquement des points 3D.

FIG. 2.1 – image avec en rouge le projetté des points 3d sur l'image

FIG. 2.2 – image avec en rouge le projetté des points 3d lampadaires sur l'image

2.2 Densifier le nuage de points

Dans cette partie nous étudierons les techniques misent en place pour densifier au mieux le nuage de points 3d qui souffre d'un manque de points par rapport au nombre de pixels surtout qui contient des ombres LIDAR.

La première étape fut de faire une segmentation radiométrique de l'image. Pour réaliser cette étape nous nous sommes servis d'une application développée au sein de TELECOM-Paristech. Celle-ci me donna de bons résultats même si l'on ne pouvait attendre un résultat parfait.

FIG. 2.3 – Image de segmentation

Deux optiques s'offraient alors à nous.

– 3D -> IMG :La première était d'estimer un point 3D comme par exemple le milieu de deux points 3D et le projeter sur l'image, s'il appartient à la même zone de segmentation, alors on garde le point 3D. Cette méthode malgré sa simplicité ne permettait pas une complétion sûre des données.
– IMG -> 3D :L'autre optique fut simple à concevoir, car il est le principe dual de la première optique. Mais me posa de nombreux problèmes quand à son développement. Le principe repose sur le travail par zone, en effet on souhaite dans cette partie estimer un ou plusieurs plans correspondant aux points 3D se projetant sur la zone et ensuite d'y projeter tous les points de la zone de segmentation.

C'est cette dernière optique que nous avons conservé car elle offrait une plus grande maniabilité des algorithmes à implémenter comme l'estimation des plans et permettait surtout de nous assuré de la complétion des données sur chaque zone traitée.
Une fois ce choix fait et l'étape de segmentation effectuée, la deuxième étape était de projeter tous les points 3D sur l'image. Cette étape est rendu triviale grâce aux fichiers de configuration en xml fourni

avec chaque photo par l'IGN. Ceux-ci comportent toutes les informations nécessaires : la valeur 3D des coins de l'image, la longueur focale, le PPS[1], le PPA[2]. De plus des programmes sont fournis et permettent d'obtenir directement des coordonnées images à partir de points 3D. Ces programmes furent modifiés pour s'intégrer à des routines plus complexes permettant directement, à partir d'un nuage de points 3D d'une rue et d'une image de cette même rue, de produire des fichiers de points correspondants aux points 3D se projetant dans l'image et des coordonnées correspondantes placées dans le repère image. Une fois ces étapes effectuées on utilise l'image binaire qui est produite durant l'étape de segmentation pour pouvoir traiter chaque zone de segmentation indépendamment.
Sur chaque zone on compte le nombre de points 3D qui se projette sur la zone, si celui-ci est supérieur à neuf, on travaille sur la zone ; sinon elle est ignorée et ne sera ainsi pas traitée (cette partie de l'image apparaîtra donc en noire sur l'image de profondeur). Si le nombre de points 3D correspondant à cette zone n'est pas suffisant, on ne peut estimer de manière correcte les plans passant par les points, les données produites ne seront pas de bonne qualité. On se limite donc aux zones où le nombre de points est suffisant.
Une fois la zone acceptée, on estime alors le ou les meilleurs plans passant par les points 3D correspondants. Cette estimation se passe en deux temps :

– Tout d'abord nous appliquons un algorithme de type RANSAC ou LMedS pour supprimer les points de la zone qui ne correspondent pas au meilleur plan, au sens de RANSAC, c'est à dire celui qui contient dans une zone tampon le plus de points.
– Sur les points qui ont été choisis, on cherche le meilleur plan possible par une minimisation par moindre carré de la longueur des points au plan.

La distance d'un point au plan est calculer de la manière suivante :

$$d = \frac{|ax + by + cz + d|}{\sqrt{a^2 + b^2 + c^2}}$$

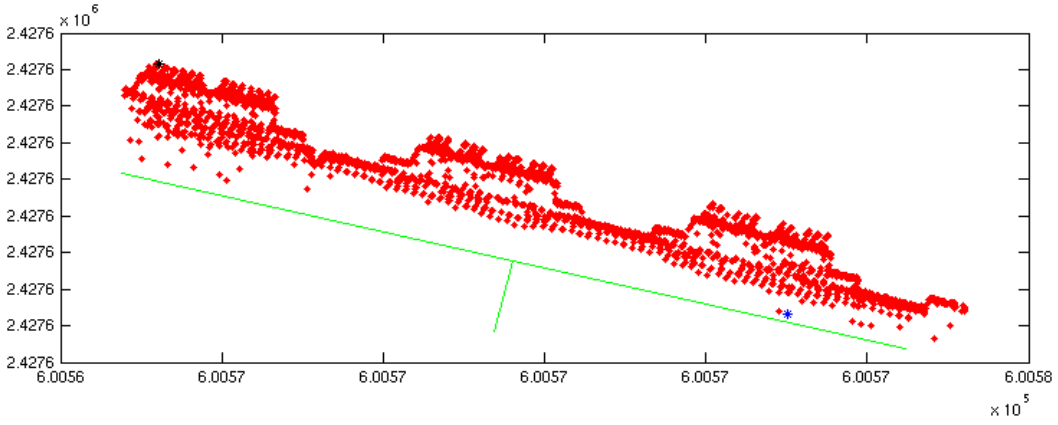
Cette partie est simplifiée en n'appliquant pas directement un algorithme de minimisation par moindre carré. On considère que le vecteur directeur de la droite est le vecteur propre associé à la plus grande valeur propre de la matrice d'inertie.
Cette matrice, pour N points M_i, $i \in [0; N]$ est calculé comme étant :

$$I = \sum_{i=0}^{N} M_i * M_i' \tag{2.1}$$

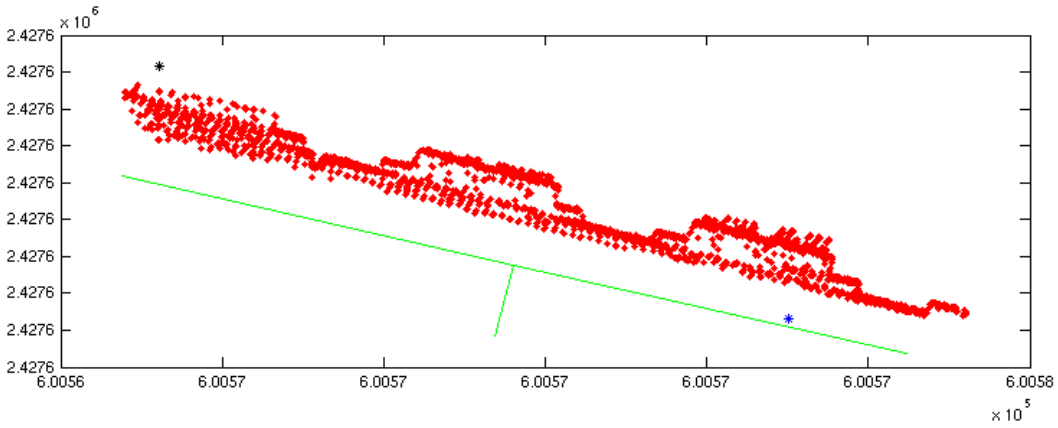
L'étape de segmentation n'est pas parfaite et certaines zones, généralement celles contenant plus de 20 points, contiennent plus d'un plan et ne peuvent donc être estimées correctement sur un plan. Ainsi les points rejetés par RANSAC, sont traités dans une deuxième phase.
Dans cette phase, on applique un RANSAC sur ces points, si la meilleure droite au sens de RANSAC contient au moins 20 points, alors on estime un nouveau plan pour ces points gardés par RANSAC. On applique cet algorithme de manière récursive jusqu'à ce que le meilleur plan ne contienne dans son tampon moins de 20 points.

Une fois ces plans estimés, commence une étape importe, celle de la reprojection. En effet maintenant chaque pixel va être reprojeté sur le plan correspondant à sa zone. Si elle contient un seul plan, alors on projette tous les pixels de la zone de segmentation sur ce plan.
Cette reprojection se fait de manière simple car une fois de plus les fichiers de configuration de l'IGN nous permettent de trouver un vecteur directeur rapidement. Cela car on récupère directement la valeur du pixel dans l'espace 3D ainsi que les coordonnées du point focale. Il est aisé de trouver le point d'intersection entre une droite dont on connaît un point et son vecteur directeur et une équation cartésienne de plan. On récupère alors un point 3D correspondant à un pixel image.

[1] Point Principal de Symétrie
[2] Point Principal d'autocolimation



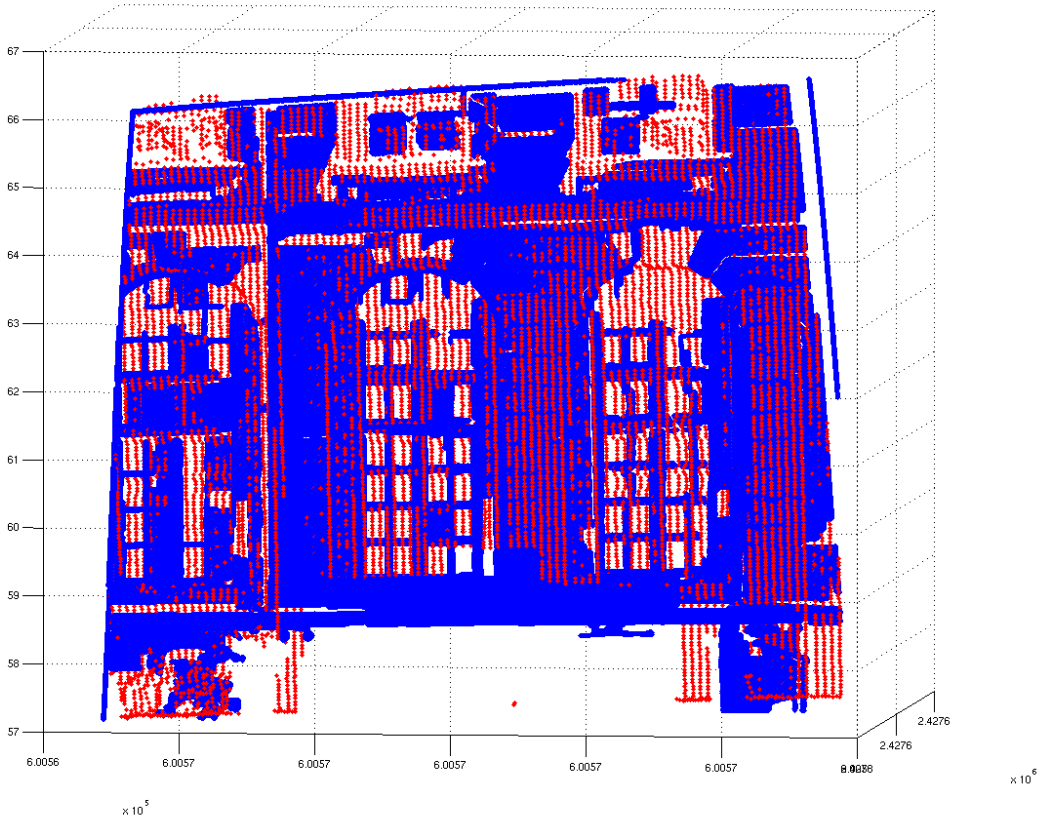
(a) Avant RANSAC



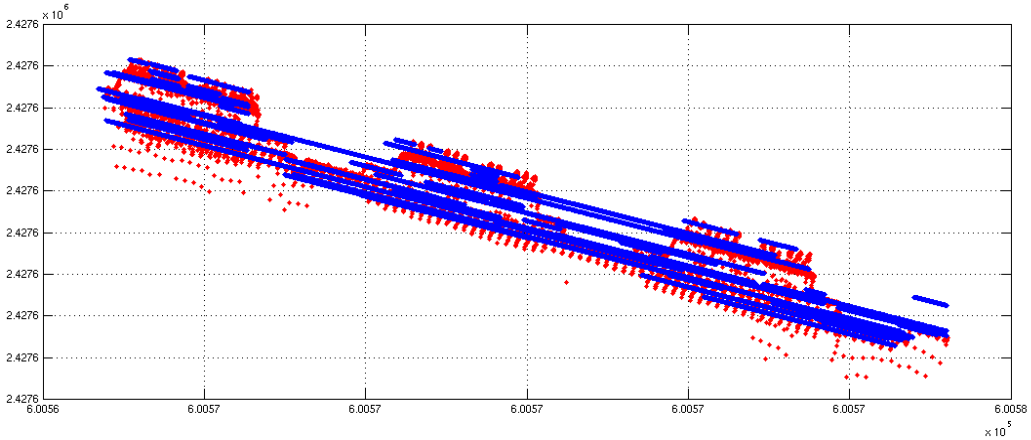
(b) Après RANSAC

FIG. 2.4 – utilisation de RANSAC sur l'ensemble des points de la façade, en coupe du dessus

On peut voir le résultat sur une coupe de face et de dessus des points rajoutés ici en bleu et des points originaux en rouge.



(a) coupe de face



(b) coupe du dessus

FIG. 2.5 – ajout des points avec en bleu les points ajoutés et en rouges les données LIDAR

2.3 Carte de profondeur et inpainting

Dans cette partie nous parlerons des étapes de création de la carte de profondeur. A la fin de l'étape de densification, nous avons un fichier de points 3D correspondants à chaque pixel de l'image. Or ces données ne permettent pas de calculer les valeurs de la carte de profondeur, pour la créer il faut ce que nous appèlerons le plan 0. C'est ce plan qui nous servira de référence. En effet dans une carte de profondeur, la valeur du pixel correspond à la distance euclidienne des points 3D à ce plan 0. Pour estimer ce plan nous nous sommes servi de la même technique que nous avions utilisée précedamment, seulement pour des raisons de logiques et de complexité nous avons projetté tous les points au sol, sur le plan $z = 0$. Nous avons alors effectué un RANSAC sur l'ensemble des points comme le montre la figure 2.4 et sur ces points nous n'avons pas chercher la meilleure droite mais la meilleure normale. En effet, lors du calcul de la matrice d'inertie il suffit de prendre le vecteur propre associé à la plus petite valeure propre pour trouvé la normale.
Il faut alors calculer la distance euclidienne de tous les points à une droite dont la normale est celle calculé précédemment au centre d'inertie du nuage de points. Il suffit après de se placer au point donnant la valeur la plus petite et inférieur à 0. Ainsi on place notre droite au point le plus "proche" de l'objectif de la caméra.

Nous trouvons ainsi l'équation cartésienne du plan 0. Nous calculons ensuite la distance euclidienne

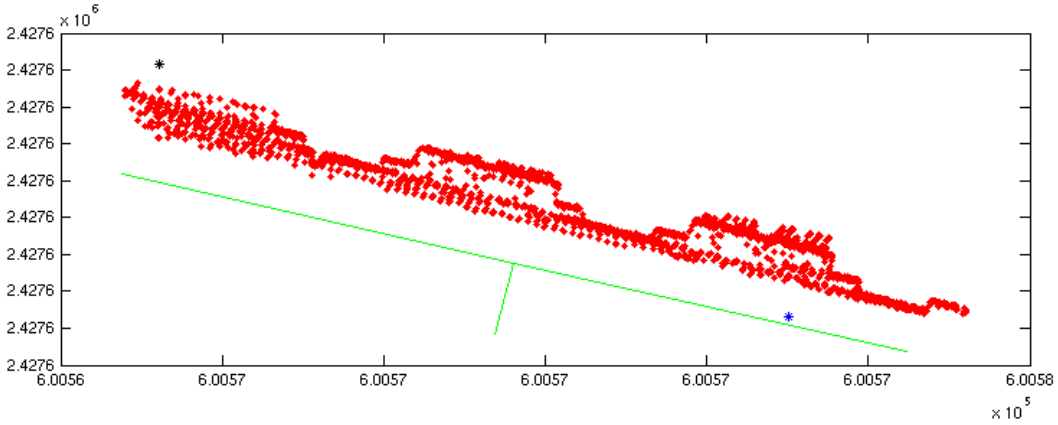


FIG. 2.6 – Résultat du calcul du plan 0 ici en vert

de tous les points à cette droite, ce qui nous permet d'avoir des valeurs de distance. Celles-ci devrons être transformées en valeur de pixel par la suite.
Pour cela nous nous servons d'une échelle de valeur. Pour que le résultat soit plus compréhensible, nous avons choisi la convention intuitive pour laquelle les points les plus éloignés sont plus sombres que les plus proches.

Ainsi, le point le plus proche prendra comme valeur 255 quand le plus éloigné prendra la valueur 1, le 0 étant réservé aux points qui n'ont pas été attribués. Ainsi il suffit d'échelonner la valeur des pixels entre le point le plus proche et le plus éloigné.
La valeur d'un pixel P_i est calculé de la manière suivante :

$$V_i = 255 * (\frac{Dist_{max} - d_i}{Dist_{max} - Dist_{min}}) \tag{2.2}$$

Une fois ce calcul effectué pour tous les pixels, il suffit d'afficher le résultat, et de le sauvegarder comme une image en niveau de gris.

FIG. 2.7 – Carte de profondeur de la façade

2.4 Problèmes rencontrés et méthodes de résolution

Dans cette partie nous expliquerons les problèmes rencontrés et comment nous avons résolu ces problèmes. Nous expliquerons aussi les limites des méthodes utilisées et les problèmes non-encore résolus. En effet dans les parties précédentes, nous n'avons montré que les résultats obtenus sans jamais mettre l'accent sur les problèmes rencontrés. Or de nombreuses difficultés ont dut être surmontées pour pouvoir arriver à ces résultats. Nous tenterons ici d'expliquer les problèmes rencontrés dans les deux parties.

2.4.1 problèmes de l'inpainting image

La première difficulté fut de surmonter la sensibilité aux paramètres expliquée dans le châpitre 1 :inpainting image où de nombreux paramètres interviennent, comme la grosseur du patch ou la taille du dictionnaire de recherche. Ces paramètres sont de véritables problèmes, car si des paramètres me donnaient de très bons résultats sur une image, elle avait tendance à déteriorer le résultat sur une autre image.
Ce problème fut résolu en acceptant un compromis entre : un résultat moyen mais marchant dans la pluspart des cas ou un très bon résultat pour un exemple. Un autre problème et qui est visible sur la figure 1.20 est le problème des structures non redondantes comme des motos ou des piétons. ce problème ne peut être résolu car normalement siu tout était parfait nous serions cencé inpainter ces structures, donc nous n'avons pas chercher à résoudre ce problème.

Dans les algorithmes qui furent développés,seuls les problèmes de programmations ont dut être surmontés. Nous avons choisi de partir sur la structure du programme de Yann Gousseau, le travail qu'il avait fournit était de bonne qualité en terme de résultat mais la complexité du code était de ce fait fortement augmentée. Ainsi l'utilisation du C, langage de programmation pour lequel j'ai été formé, m'a posé des problèmes tant la manière de programmer était différente de celle que l'on m'a enseigné. L'élaboration de ce code m'a demandé enormément de rigueur, il était si facile de se tromper.
Pour exemple, un "bug" qui me prit près de deux jours à être résolu était lié à une inversion de symbole entre l'article et le code. Ainsi en regardant le nom je me suis imaginé qu'il s'agissait d'un paramètre particulier, en fait il s'agissait d'un autre. C'est en cherchant dans la documentation fournit que je me suis rendu compte qu'il avait changé les dénominations.

Le seul algorithme qui me posa réellement des problèmes fut celui de l'ICM. La mise en place de l'algorithme ne fut pas simple, beaucoup d'inconpréhensions ont ralenti sa mise en place et de nombreuses fois je dus me rendre auprès de mon maitre de stage pour avoir des précisions ce qui ralenti le développement.

2.4.2 Problèmes de l'inpainting 3D

Le seul problème rencontré fut la mise ne place du densifieur de points 3D. La carte de profondeur ne fut quand à elle qu'une simple application des différents algorithmes mis en place dans le densifieur.

Le premier problème fut l'utilisation de l'image binaire de segmentation. Ce problème fut résolu par l'utilisation d'un morceau de code développé par Jean-Marie Nicolas et disponible gratuitement sur son site et libre de droit.
Un autre problème fut la mise en place du calcul de la matrice d'inertie et des valeurs propres. Deux options s'offraient à moi, soit reprogrammer une fonction calculant les vecteurs et valeurs propres ou utiliser des fonctions d'ors et déjà programmées. Mon choix se porta sur la deuxième solution. Restait à savoir quelle fonction utiliser, ce choix fut vite résolue, car je connais la GSL[3], qui propose de très

[3]GNU Scientific Library

bonnes fonctions mathématiques qui ont été soumises de très nombreuses fois aux regards critiquess d'un public de connaisseurs et ont prouvé depuis longtemps leur efficacité.
Enfin des problèmes de géométrie dans l'espace m'ont ralenti, ceux-ci étaient dignes d'un contrôle de lycée mais mes notions de géométrie étaient quelques peu anciennes. Fort heureusement, de nombreuses pages internet ont permis à ces souvenirs de revenir plus vite que je ne le pensais.

Enfin encore une fois des problèmes de programmation ralentirent le développement de ces applications. De nombreux bugs existaient et ma gestion mémoire était catastrophique. Un exemple, au lieu de traiter chaque point et de regarder s'il convenait, je stockais tout dans des variables dont la mémoire n'était libérée nulle part. Plusieurs jours passés sur des debugueurs ou profileurs comme `gdb`, `valgrind` ou `electric fence` m'ont permis de régler ces problèmes. Ainsi le résultat final est un programme rapide, efficace, robuste et qui ne souffre d'aucune fuite mémoire.

Conclusion

Nous avons dans ces deux parties comment nous avons tenté de mettre en place des solutions aux deux problèmes qui m'ont été posés : l'inpainting des images et l'inpainting 3D.
Dans le premier cas nous sommes allé loin dans la recherche de nouveaux algorithmes améliorant les résultats. Mais durant notre travail nous avons put soulever deux limites majeures aux solutions proposés :

– La sensibilité aux paramètres, qui peut être gênant si on veut automatiser l'algorithme
– L'algorithme atteint ses limites si l'image manque de redondance, c'est à dire que l'algorithme a dut mal à fonctionner si la structure que nous traitons n'apparait pas plus de deux fois.
– On peut aussi rajouter que si la majorité de l'image est occluse, l'algorithme trouve peu d'information sur l'image et donne alors de très mauvais résultats.

Plusieurs possibilités s'offrent à nous pour résoudre ces problèmes, tout d'abord dans le cas d'une occlusion trop importante ou pas assez redondante, on peut imaginer passer par du mosaïcage d'image pour arriver à nous donner plus d'information. Ceci rendu possible que s'il on possède un jeu d'images. De même pour la sensibilité aux paramètres, des paramètres par défaut ont été cherché. On cherchais ainsi des paramètres ne donnant pas pleinement satisfaction mais qui marchait sur un maximum d'images. Ces paramètres ont été trouvés et le programme implémente ces résultats, ainsi si on ne lui spécifie pas de paramètres particuliers ceux trouvés sont alors utilisés.

Dans l'inpainting 3D, notre problème repose principalement sur le manque de recul que nous avons sur l'algorithme. En effet ce fut seulement peu de temps avant la fin que des résultats probants sont venus appuyer nos théorie. Malheureusement nous n'avons pas eu le temps de le téster sur d'autres images (notamment sur l'image du lampadaire), et de plus nous n'avons pas eu le temps de tester l'inpainting image sur ces cartes de profonfeur. Ainsi il reste une phase de test à terminer pour définitivement assoir notre théorie selon laquelle on peut créer des points 3D qui sont masqué par l'inpainting d'une carte de profondeur.

Bibliographie

[1] Antonio Criminisi, Patrick Pérez et Kentaro Toyama,
Region Filling and Object Remouval by Exemplar-Based Image Inpainting,2004

[2] Alexi Efros et Thomas Leung,
Texture Synthesis by Non-Parametric Sampling

[3] Patrick Perez, Michel Gangnet et Andrew Blake,
Patchwork : Example-Based Region Tiling for Image Editing,2004

[4] Simon Masnou et Jean-Michel Morel ,
Level Lines Based Disocclusion,1998

[5] A.R.Dick, P.H.S.Torr et R.Cipolla,
Modelling and Interpretation of Architecture from several Images,2002

[6] Pascal Müller, Gang Zeng, Peter Wonka,Luc Van Gool,
Iamge-Based Procedural Modelling of Facades,2007

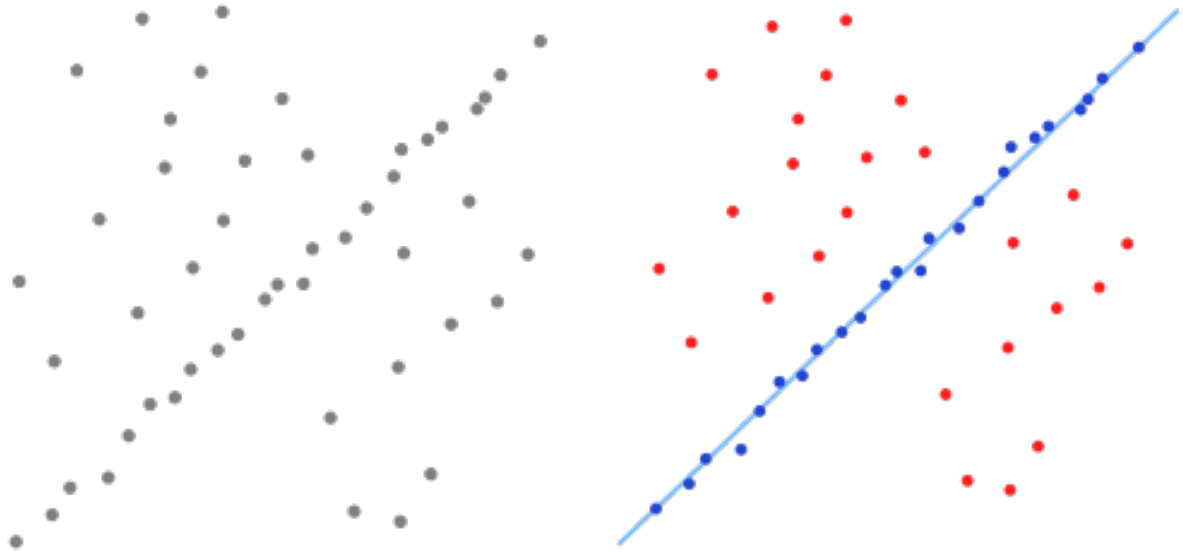
[7] V.Do, G.Lebrun, L.Malapert, C.Smet et D.Tschumperlé,
Inpainting d'Images Couleurs pas Lissage Anisotrope et Synthèse de Textures , Laboratoire GREYC

RANSAC

C'est une méthode d'estimation d'un modèle mathématique contenant des points non désirés appelés "outliers". C'est un algorithme non-déterministe, du point de vue ou la complexité et le nombre d'itération dépend de la sensibilité des paramètres. L'algorithme a été publié la première fois par Fischler et Bolles en 1981.

L'hypothèse de départ est que tout jeu de données contient des points qui suivent un modèle et que l'on appelle "inliers" et des points incohérents appelé "outliers" qui peut correspondre à du bruit.

Exemple



(a) jeu de points (b) Sorti de RANSAC :deux jeux de points

FIG. 8 – Exemple d'utilisation de RANSAC

Nous insérons ici une représentaion de l'algorithme tel que nous l'avons utilisé :

Algorithme 1 RANSAC sur un jeux de données 3D

ENTRÉES: 3d data set, mathematical model, λ as the threshold, n maximum number of iteration,N number of points

SORTIES: 3d data set of inliers

$i \leftarrow 0$
$nbmaxpt \leftarrow 0$
tant que $i < n$ **faire**
  calcul d'equation de droite Δ_i pour un couple de point aléatoire
  $j \leftarrow 0$
  tant que $j < N$ **faire**
    $d(j, \Delta_i)$ avec $d = SSD$
    si $d(j, \Delta_i) < \lambda$ **alors**
      $cpt++$
    fin si
  fin tant que
  si $cpt > nbmaxpt$ **alors**
    $nbmaxpt \leftarrow cpt$
    $\Delta_{fin} = \Delta_i$
  fin si
fin tant que

tant que $j < N$ **faire**
  $d(j, \Delta_{fin})$
  si $d(j, \Delta_{fin}) < \lambda$ **alors**
    $inlers \leftarrow j$
  fin si
fin tant que

Calcul de l'intersection droite/plan

Dans cette partie nous expliquerons comment nous avons calculer le point d'intersection entre une droite dont nous connaissons deux points et un plan dont nous connaissons l'équation cartesienne. Soit $A = (x_a, y_a, z_a)$ et $B = (x_b, y_b, z_b)$ les deux points de la droite et P le plan d'équation $a \cdot x + b \cdot y + c \cdot z + d = 0$ et soit $M = (x_m, y_m, z_m)$ le point d'intersection.

On sait que $M = A + k \cdot \overrightarrow{AB}$
Ainsi :

$$M = \begin{cases} x_m = x_a + k \cdot (x_b - x_a) \\ y_m = y_a + k \cdot (y_b - y_a) \\ z_m = z_a + k \cdot (z_b - z_a) \end{cases}$$

Si on remplace dans l'équation de la droite et qu'on place en facteur k :

$$k = -1 * \frac{a \cdot x_a + b \cdot y_a + c \cdot z_a + d}{a \cdot (x_b - x_a) + b \cdot (y_b - y_a) + c \cdot (z_b - z_a)}$$

Ainsi une fois k calculé il suffit de remplacer dans le système précédent et on obtient les coordonnées du point d'intersection.
On peut appliquer cette algorithme à la projection des nouveaux points images, mis dans le référentiel 3D, sur un plan estimé aussi bien que la projection d'un point 3D sur le plan image.