Mémoire de stage de L3 : Panoramas

Stagiaires : Julie Gauthier et Samy Jaziri
Encadrants : Corinne Vachier et Loïc Simon
ENS de Cachan, CMLA

2 juillet 2012

Panoramix : « C'est une bonne situation ça, scribe ? ... »
Misenplis : « Oh ! C'est une situation assise... Accroupie, plutôt. »

Résumé

On présente une méthode pour construire de manière automatique un panorama à partir de deux photos. Une des deux photos est déformée selon une homographie. Pour trouver une homographie qui convient, on détecte et on apparie les points particuliers de chaque photo grâce à l'algorithme SIFT que nous n'expliquons que succinctement dans ces notes. On trouve une homographie par la méthode RANSAC. Là où les photos se superposent, on a utilisé deux méthodes pour calculer la couleur des pixels. La première est assez simple : elle consiste à faire la moyenne entre les deux photos. Cependant, cette méthode fait apparaître deux types d'artefacts dont nous discuterons. La seconde méthode utilise à bon escient les équations de Poisson et donne des résultats bien meilleurs.

Abstract

A method to generate automatically panoramas from two photos is introduced. One of the photos is deformed by a homography. To find a suitable homography, particular points of each photos are detected and matched with the SIFT algorithm that won't be presented in details here. Then, an homography is found by the RANSAC method. Where the photos overlap, two methods were used to compute the pixels color. The first is naive : it consists in averaging the two photos. Nevertheless, this method is affected by two kinds of artefacts discussed in this notes. The second method build upon the Poisson equations yields much better results.

Chapitre 1

Aspect théorique

1.1 Position du problème

Certains photographes souhaitent réaliser des vues panoramiques de paysage. Or, en collant à la main les photos prises les unes après les autres, le panorama obtenu ne paraît pas naturel, comme on le voit sur la figure 1.1. Il faut déformer les photos pour que le résultat soit réaliste, pour que les photos se collent du mieux possible l'une sur l'autre. On souhaiterait obtenir des résultats similaires à celui de la figure 1.2.

De plus, il est intéressant de rendre ce processus totalement automatique, de telle sorte que les photographes ne perdent pas de temps à traiter les photos avant d'en faire des panoramas.

Ainsi, on souhaite écrire un programme qui à partir uniquement de la donnée de deux photos, réalise un panorama à partir de ces deux photos. Notre programme devra positionner et déformer les photos sans intervention de l'utilisateur. Des solutions à ce problème ont déjà été proposées, comme par exemple celle de Brown et Lowe dans [1] et [2].

Nous avons souhaité mettre nous-même en œuvre des méthodes pour résoudre ce problème.

L'idée générale de notre programme est de reconnaître des points particuliers sur chacune des photos, de les apparier et de trouver quelle transformation effectuer sur une des deux images pour qu'en les superposant, les points appariés se confondent d'une photo à l'autre.

1.2 Géométrie projective et homographie

On ne peut pas choisir n'importe quel type de transformation pour passer d'une image à l'autre. Comme sur des photos les droites de l'espace sont représen-

(a) Photo d'origine 1

(b) Photo d'origine 2

(c) Panorama

FIGURE 1.1 – Panorama réalisé à la main

FIGURE 1.2 – Panorama réalisé par notre algorithme

tées par des droites dans le plan de la photo, nous avons choisi de conserver cette propriété. La transformation qu'on effectuera sur notre image conservera l'alignement des points.

Il est à noter que ce n'est pas la seule méthode possible. On peut trouver des panoramas où les droites de l'espace ne sont pas représentées par des droites

dans le plan du panorama, comme par exemple à la fin de l'article [1].

1.2.1 La géométrie projective : un point de vue adapté au problème

Une photo peut être vue comme une portion de $\mathbb{R}^2$. Mais ce n'est pas le point de vue le plus adapté à ce que nous allons faire. On va travailler dans l'espace projectif.

On choisit un modèle simplifié pour l'appareil photo présenté par la figure 1.3. On dit que lorsqu'on prend une photo, l'appareil réalise une projection centrale de centre O le centre optique de l'appareil photo de l'espace sur le plan $\mathcal{P}$ de la photo (aussi appelé plan rétinien), c'est-à-dire que l'image d'un point P de l'espace sur la photo est l'intersection de la droite (OP) avec le plan $\mathcal{P}$. On doit avoir $O \notin \mathcal{P}$.

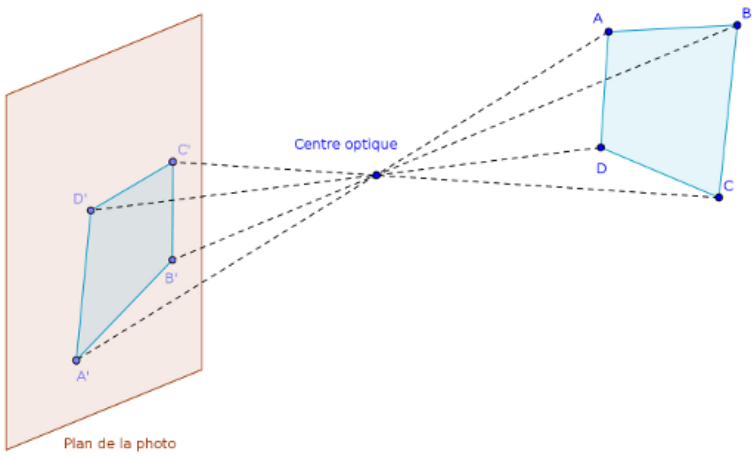


FIGURE 1.3 – Modèle de projection centrale réalisée par un appareil photo

Remarque 1.2.1. Les points du plan parallèle à $\mathcal{P}$ passant par O n'ont pas d'image par cette projection centrale.

Remarque 1.2.2. Tous les points d'une droite passant par O ont la même image par une projection centrale.

Propriété 1.2.1. *Une projection centrale conserve :*
- *l'alignement dans l'ordre,*
- *la concourance des droites.*

Une projection centrale ne conserve pas le parallélisme : les images de deux droites parallèles distinctes sont des droites sécantes lorsque ces droites ne sont pas parallèles à $\mathcal{P}$.

Ce phénomène est à l'origine de la définition des espaces projectifs $\mathbb{P}^2$ et $\mathbb{P}^3$ à partir des espaces $\mathbb{R}^2$ et $\mathbb{R}^3$ respectivement où toutes les droites se coupent. En particulier, les droites parallèles se coupent en des points dit idéaux ou à l'infini.

Remarque 1.2.3. Cette notion correspond aux points de fuite en peinture.

On va définir clairement $\mathbb{P}^2$ car c'est surtout cet espace que l'on sera amené à utiliser. Pour $\mathbb{P}^3$, l'idée est la même. On pourra trouver une présentation de cet espace dans la partie 0 de [3].

Les points de $\mathbb{R}^2$ peuvent être représentés par leurs coordonnées cartésiennes $(x, y) \in \mathbb{R}^2$. De même, les points de $\mathbb{P}^2$ sont représentés par des coordonnées dites homogènes $(x, y, k) \in \mathbb{R}^{3*}$. Deux points de $\mathbb{P}^2$ de coordonnées homogènes (x, y, k) et (x', y', k') sont confondus lorsqu'il existe $\alpha \in \mathbb{R}^*$ tel que $(x, y, k) = \alpha(x', y', k')$

Si $k \neq 0$, le point de $\mathbb{P}^2$ de coordonnées homogènes (x, y, k) correspond au point de $\mathbb{R}^2$ de coordonnées cartésiennes $(\frac{x}{k}, \frac{y}{k})$.

Si $k = 0$, le point de $\mathbb{P}^2$ de coordonnées homogènes $(x, y, 0)$ est un point idéal.

Les droites de $\mathbb{R}^2$ sont habituellement représentées par leur équation cartésienne : $D = \{(x, y) \in \mathbb{R}^2 | ax + by + c = 0\}$ où $(a, b, c) \in \mathbb{R}^3$ et $(a, b) \neq (0, 0)$. On remarque que (a, b, c) est défini à un coefficient multiplicatif près, comme les points de $\mathbb{P}^2$.

De même, une droite de $\mathbb{P}^2$ peut-être représentée par son équation $D = \{(x, y, k) \in \mathbb{P}^2 | ax + by + ck = 0\}$. En fait, on peut même écrire cette équation comme un produit scalaire :

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} . \begin{pmatrix} x \\ y \\ k \end{pmatrix} = 0$$

C'est là qu'on voit que les droites et les points jouent le même rôle en géométrie projective !

Théorème 1.2.1 (Principe de dualité)**.** *À chaque théorème dans $\mathbb{P}^2$ correspond un théorème dual, obtenu en permutant le rôle des points et celui des droites.*

On a dit qu'en géométrie projective, toutes les droites distinctes se coupent en un point. Vérifions ce résultat.

Soit $(a, b, c) \in \mathbb{R}^{3*}$ et $(a', b', c') \in \mathbb{R}^{3*}$ non colinéaires. Ce sont des représentations homogènes de deux droites distinctes. Cherchons leur intersection : on a le système

$$\begin{cases} ax + by + ck = 0 \\ a'x + b'y + c'k = 0 \end{cases}$$

En fait, le vecteur (x, y, k) est orthogonal aux vecteurs (a, b, c) et (a', b', c'). Comme il est défini à une constante multiplicative près, on peut prendre le pro-

duit vectoriel :

$$\begin{pmatrix} x \\ y \\ k \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} \wedge \begin{pmatrix} a' \\ b' \\ c' \end{pmatrix}$$

Proposition 1.2.1. *Soient D et D' deux droites distinctes de $\mathbb{P}^2$. D et D' se coupent en $P = D \wedge D'$.*

Remarque 1.2.4. Si D et D' sont parallèles, alors (à un coefficient multiplicatif près que l'on écrit pas) $a = a'$ et $b = b'$. On obtient donc $k = 0$: P est un point idéal.

En appliquant le principe de dualité à la proposition 1.2.1, on obtient :

Proposition 1.2.2. *Soient P et P' deux points distincts de $\mathbb{P}^2$. Une droite D passe par P et P', et on a $D = P \wedge P'$.*

Remarque 1.2.5 (À propos de la droite à l'infini). Soient $(x, y, 0)$ et $(x', y', 0)$ deux points idéaux distincts. Une droite les relie :

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} \wedge \begin{pmatrix} x' \\ y' \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \text{ à un coefficient multiplicatif près.}$$

Remarque 1.2.6. En fait, $\mathbb{P}^2$ peut être vu comme $\mathbb{R}^{3*}$ quotienté par la relation d'équivalence $\doteq$ avec $(x, y, k) \doteq (x', y', k') \iff \exists \alpha \neq 0 | (x, y, k) = \alpha(x', y', k')$.

Pour en revenir à la projection centrale réalisée par l'appareil photo, on peut dire que si on considère l'espace affine de centre O et si on appelle O' le projeté orthogonal de O sur $\mathcal{P}$ et qu'on le choisit comme origine dans $\mathbb{P}^2$ représentant la photo (voir figure 1.3), avec $\vec{e_z} = \vec{OO'}$, les coordonnées homogènes d'un point P de $\mathbb{P}^2$ peuvent être interprétées comme des coordonnées cartésiennes d'un point de la droite (OP). Cela rejoint la remarque 1.2.2.

1.2.2 Une transformation qui conserve l'alignement

Hartley et Zisserman, dans [3], définissent une homographie comme une application $h : \mathbb{P}^2 \to \mathbb{P}^2$ bijective telle que trois points sont alignés si et seulement si leurs images par h le sont.

Remarque 1.2.7. Les homographies conservent l'alignement. Donc, par dualité, elles conservent aussi la concourance.

On a le théorème suivant :

Théorème 1.2.2. *Une application $h : \mathbb{P}^2 \to \mathbb{P}^2$ est une homographie si et seulement s'il existe une matrice $H \in \mathcal{GL}_3(\mathbb{R})$ définie à une constante multiplicative près telle que, en représentant tout point $P \in \mathbb{P}^2$ et son image $P' = h(P)$ par leurs coordonnées homogènes, on a la formule :*

$$P' = HP$$

On peut trouver la preuve de l'implication indirecte dans [3]. Par contre, on y lit aussi que l'implication directe est difficile à démontrer.

Ainsi, une homographie h est représentée par une matrice $H \in \mathcal{GL}_3(\mathbb{R})$ homogène c'est-à-dire définie à une constante multiplicative près. Par conséquent, pour déterminer h (et H) de manière unique, il faut et il suffit de connaître les images de quatre points telle qu'aucun des quatre points n'est aligné avec deux autres points.

Une personne désirant réaliser un panorama prend plusieurs photos d'affilée, en faisant tourner l'appareil photo sur lui-même. Du moins, c'est le modèle que l'on choisit. Cela correspond à laisser O invariant d'une photo à l'autre et à faire tourner $\mathcal{P}$ autour d'un axe passant par O. On va montrer que la transformation pour passer d'une photo à l'autre est une homographie. En fait, changer $\mathcal{P}$ sans changer O d'une photo à l'autre revient à réaliser une homographie.

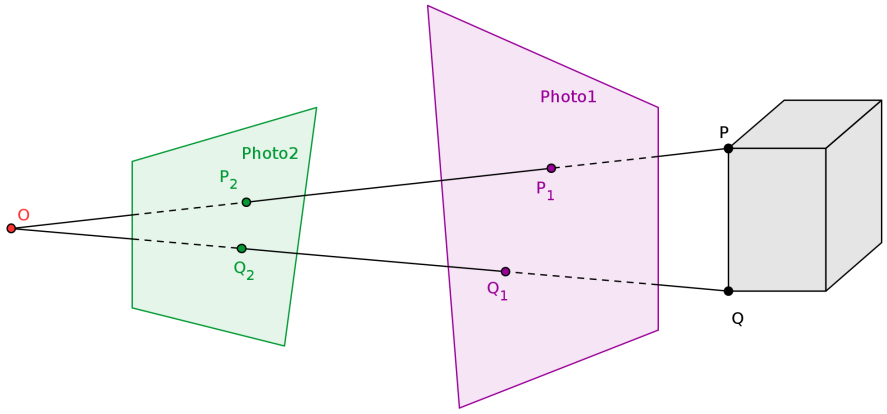


FIGURE 1.4 – Est-ce une homographie ?

La figure 1.4 présente un cas plus général : d'une projection à l'autre on change le plan de la projection mais pas son centre. Soit f la projection de centre O du premier plan rétinien $\mathcal{P}_1$ sur le second $\mathcal{P}_2$. On peut voir f comme une application de $\mathbb{P}^2 \to \mathbb{P}^2$, f permet de passer de la première photo à la seconde. En fait, soit P un point de l'espace, on appelle P_1 son image sur la première photo, et P_2 son image sur la seconde photo. On sait que $P_1, P_2 \in (OP)$. Ainsi,

P_2 est à la fois l'image de P et de P_1 par la projection de centre O et de plan rétinien $\mathcal{P}_2$. Par conséquent, f permet bel et bien de passer d'une photo à l'autre.

Pour montrer que le passage d'une photo à l'autre est une homographie, il suffit de montrer que f est une homographie, c'est-à-dire est bijective et conserve l'alignement.

On sait que l'image d'une droite par une projection centrale est une droite (dans le cas général) et un point si et seulement si la droite en question passe par O. Or, aucune droite de $\mathcal{P}_2$ ne passe par O car $O \notin \mathcal{P}_2$, donc l'image par f d'une droite est une droite : f conserve l'alignement.

Soit g la projection de centre O de $\mathcal{P}_2$ sur $\mathcal{P}_1$, on voit que sans difficulté que $f \circ g = \mathrm{Id}_{\mathbb{P}^2}$. Donc f est bijective.

Donc, f est une homographie : le passage d'une photo à l'autre est une homographie.

Reste à trouver laquelle !

1.3 Algorithmes pour le choix de l'homographie

Comme on vient de le voir, il nous faut quatre points et leurs images pour déterminer une homographie.

1.3.1 Scale-Invariant Features : un algorithme pour trouver des points particuliers et les apparier

Pour détecter et apparier les points particuliers sur chacune des photos, nous avons utilisé l'algorithme présenté par D. G. Lowe dans [4]. Nous ne l'avons pas codé nous même mais avons utilisé le code `Matlab` proposé par A. Vedaldi et B. Fulkerson dans [5]. Nous avons en particulier utilisé les fonctions :

- `vl_sift` qui détecte les points particuliers et calcule leurs descripteurs,
- `vl_ubcmatch` qui apparie les points particuliers en fonction de leurs descripteurs.

Cet algorithme détecte des points particuliers et leur associe des descripteurs qui décrivent la particularité de chaque point. Ainsi, pour apparier les points d'une photo à l'autre, on regarde si les descripteurs sont proches, c'est-à-dire si la distance d'un descripteur à l'autre multipliée par un seuil (`tresh`) est inférieur à la distance du premier descripteur à tous les autres (voir [5]). Dans ce cas, on les apparie.

1.3.2 RANdom SAmple Consensus : un algorithme pour calculer une homographie convenable

Maintenant que les points particuliers sont détectés et appariés, on va calculer une homographie qui convient. Pour cela, nous avons utilisé l'algorithme RANSAC implémenté en `Matlab` par E. Wigging dans [6], en particulier la fonction `findHomography`. L'algorithme RANSAC est décrit précisément dans [7].

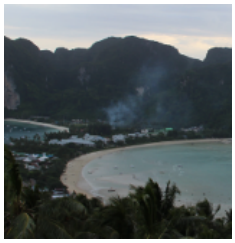
On présente l'algorithme dans le cas présent. Comme données, on a un grand nombres de points particuliers de la deuxième photo, et ce qui semble être leurs images sur la première photo. On cherche une homographie qui appliquée à la première photo, fait correspondre les points particuliers par paires.

On sait qu'il nous faut quatre points et leurs images pour déterminer de manière unique une homographie. On va donc sélectionner aléatoirement un ensemble de quatre points et calculer l'homographie h qui y correspond. Puis on applique h aux autres points dont l'image est connue. Si on ne tombe « pas trop loin » des points attendus, on considère que la paire testée est valable pour le modèle h (inliers en anglais), sinon, on considère qu'elle ne l'est pas (outliers). S'il y a « suffisamment » de paires valables, le modèle est considéré correct. Parmi les modèles corrects, on sélectionne le « meilleur » en ne considérant que les paires valables pour tous les modèles corrects, et on calcule l'homographie h correspondante (sous une de ses formes matricielles H).

Cette méthode permet d'éliminer les effets des paires qui ont été formées par l'algorithme SIFT mais qui ne correspondent pas à des paires dans la réalité.

Remarque 1.3.1. On doit définir des mesures pour savoir ce que « pas trop loin », « meilleur » et « suffisamment » veulent dire. Par exemple, pour « pas trop loin »pour une paire (P_1, P_2), on peut choisir la distance $P_1h(P_2)$. Attention, on ne sait pas si c'est le cas ici, la documentation pour [6] étant trop succincte.

Il est à noter que ce processus est aléatoire. Ainsi, pour deux photos, l'homographie calculée n'est pas toujours la même. C'est le cas surtout avec des photos d'origine de basse résolution, comme on peut le voir sur la figure 1.5.



(a) Photo d'origine 1

(b) Photo d'origine 2

(c) Premier résultat

(d) Second résultat

FIGURE 1.5 – Deux panoramas obtenus avec les mêmes images d'origine par le même algorithme

Chapitre 2

Mise en œuvre

Dans un premier temps, nous étions deux à travailler. On a pu programmer en C++ car Samy connaissait déjà le langage. Quand il est parti, Julie a repris le travail en `Matlab`, langage qui était plus accessible [1].

Rappelons le principe de l'algorithme.
En entrée : deux images `img1` et `img2`.
En sortie : une image, le panorama `pan` obtenu en appliquant la bonne homographie à `img1` pour qu'elle se superpose sur l'image `img2`.
Algorithme :

1. trouver et apparier les points particuliers de `img1` et `img2` (SIFT),
2. trouver une homographie `H` qui convient (RANSAC),
3. appliquer `H` à `img1`, on appelle `Himg1` le résultat,
4. superposer `Himg1` et `img2`, on appelle `pan` le résultat,
5. renvoyer `pan`.

On a déjà vu comment faire les étapes 1 et 2. Ici, on détaille les étapes 3 et 4.

2.1 À propos du ré-échantillonnage (étape 3)

Lorsqu'on applique `H` à tous les pixels de l'image `img1`, deux phénomènes se produisent. On les voit sur la figure 2.1. D'une part `Himg1` « dépasse » de la grille sur laquelle on avait `img2`. D'autre part, les pixels de `img1` ne tombent pas exactement sur des pixels de la grille correspondant à `img2`.

Le premier problème se règle en agrandissant et en translatant la grille correspondant à `img2`.

1. pour une fille, comme dirait Loïc Simon !

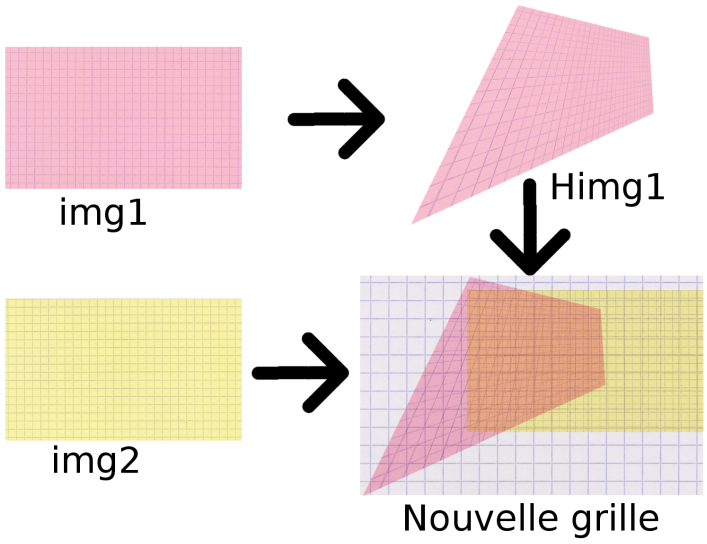


FIGURE 2.1 – Schéma représentant l'étape 3

Pour le second problème, on considère les pixels de la nouvelle grille, et on interpole leur couleur à partir `img2` et `Himg1`. Pour cela, on procède à une interpolation linéaire, en utilisant la fonction `TriScatteredInterp` en `Matlab`.

On obtient ainsi `img2bis` et `Himg1bis` qui correspondent bel et bien à la grille sur laquelle on coloriera `pan`.

2.2 À propos de la construction du panorama (étape 4)

La question qui se pose ici est la suivante : comment faire exactement l'étape 4 de l'algorithme ?

2.2.1 Comment réaliser le collage ?

Notre idée consiste à réaliser un `canevas` servant à guider la construction du panorama `pan`. Le `canevas` a la même taille que `pan`, et est colorié avec des `1` là où seule l'image `Himg1bis` doit être « collée », avec des `2` là où on ne doit mettre que les pixels de `img2bis`, des `3` là où on doit superposer les deux images, et des `0` partout ailleurs. Pour ce faire, on calcule `canevas1` un canevas pour `Himg1bis`, en calculant l'endroit où sont projetés les coins de `img1`, en les reliant (fonction `trace_segment`, cf annexe), et en coloriant avec des `1`

l'intérieur du quadrilatère ainsi obtenu (fonction `quadrilatere_plein`, cf annexe). On fait de même avec `img2bis`, et on obtient `canevas2`. Puis, on pose `canevas=canevas1+2*canevas2`. La figure 2.2 montre un exemple de canevas obtenu.

Les zones `0`, `1` et `2` sont remplies sans problème. En ce qui concerne la zone `3`, on doit mélanger `Himg1bis` et `img2bis`. Pour cela, nous avons mis en place deux méthodes.

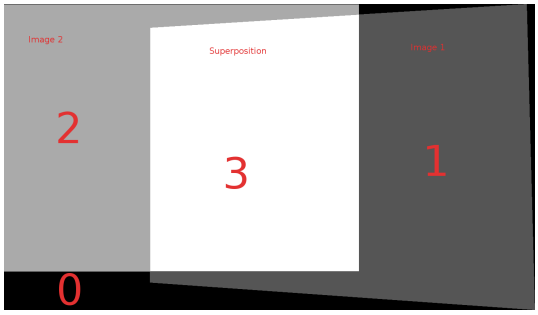


FIGURE 2.2 – Canevas

2.2.2 Méthode naïve : faire la moyenne des deux images

La première méthode est simple : on fait la moyenne entre `Himg1bis` et `img2bis`. Cette méthode donne de beaux résultats lorsque les photos ont été prises au même moment, quand l'éclairement est le même d'une photo à l'autre. Le panorama de la figure 2.3 a été réalisé à partir de trois photos prises en début de soirée à Ko Phi Phi (Thaïlande).

Cependant, les résultats ne sont pas toujours aussi bon : le panorama de la figure 2.4 a été créé à partir de deux photos prises vers midi. On y voit [2] une discontinuité entre les deux photos aux bords de la zone `3`. On observe parfois des effets de flou, si les photos ne se superposent pas très bien, comme sur la figure 2.5.

2.2.3 En utilisant les équations de Poisson

Pour réduire les imperfections produites par la méthode naïve, nous avons mis en œuvre une autre méthode. L'article [8] présente une méthode d'édition sans effets de bords d'images en utilisant les équations de Poisson. Nous avons pensé que ce type de méthode était adapté à notre problème et l'avons testé.

2. si on a une bonne vue.

(a) Photo d'origine 1

(b) Photo d'origine 2

(c) Photo d'origine 3

(d) Panorama obtenu

FIGURE 2.3 – Ko Phi Phi, Thaïlande

FIGURE 2.4 – Ko Phi Phi, encore (la flèche a été rajoutée à la main)

L'idée est simple : la zone de l'image que l'on veut éditer est bornée et on connaît les conditions au bord. Au lieu de colorier directement, on va d'abord en prendre le gradient, puis réintégrer sur le domaine connaissant les conditions au bord. De cette façon, on est sûr de ne pas observer de discontinuité. La figure 2.7 montre un des résultats que nous avons obtenu.

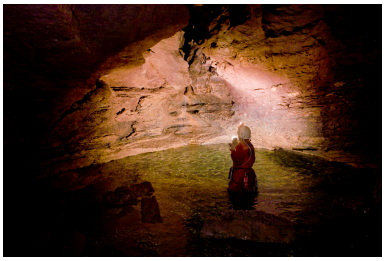
FIGURE 2.5 – Flou

FIGURE 2.6 – Résultat obtenu à partir des photos 1.1a et 1.1b : c'est encore insuffisant par rapport à la figure 1.2 !

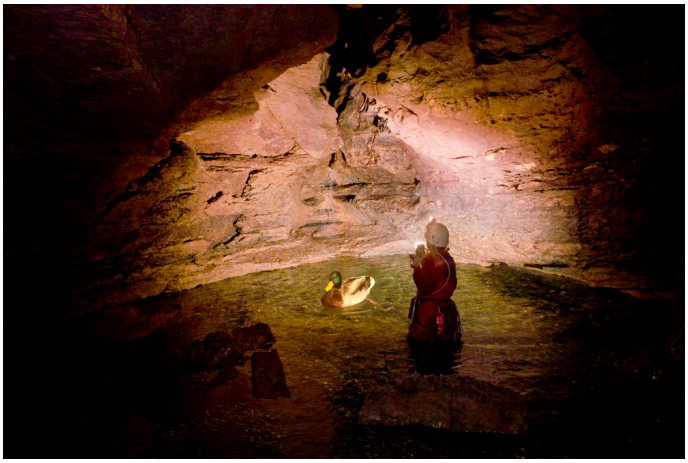
Dans un second temps, nous avons adapté ce processus au cas des panoramas, mais quel gradient prendre ? La figure 2.8 montre plusieurs résultats.

Ici, contrairement au cas de l'incrustation, il n'est pas logique de privilégier une image par rapport à l'autre en prenant son gradient. Par conséquent, nous avions d'abord eu l'idée d'utiliser la moyenne des deux gradients, ce qui a donné des résultats inattendus, comme le montre la figure 2.8e. De toute façon, cette idée n'était pas très bonne, au lieu que ce soit l'image qui est flouée maintenant,

(a) Un canard



(b) Une grotte



(c) Cette image n'a pas vocation à être réaliste !

FIGURE 2.7 – Incrustation d'un canard dans une grotte

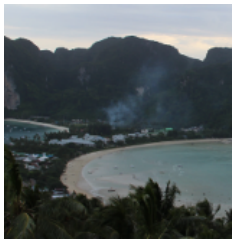
c'est son gradient, ce qui en réintégrant donne des résultats catastrophiques !

Pourtant, en prenant le gradient d'une des deux images, le panorama obtenu est réaliste (voir figures 2.8c et 2.8d). Ensuite, nous avons essayé avec celui des deux gradients qui a la plus grande norme, de cette façon, on ne privilégie pas une image par rapport à l'autre et le panorama que l'on obtient est réaliste (voir figure 2.8f).

2.3 Problèmes rencontrés

Dans cette partie, nous présentons quelques problèmes que nous avons eu pour l'implémentation, et les solutions que nous avons trouvées. Ces problèmes sont propres au langage de programmation que nous avons utilisé : `Matlab`.

Le plus gros problème auquel nous avons été confronté provient de la taille des objets que nous avons utilisés. Les images avec lesquelles nous avons travaillé, par exemple celles de la figure 2.3, contiennent environ $1000 \times 500 =$



(a) Photo d'origine 1

(b) Photo d'origine 2

(c) En prenant le gradient de `Himg1bis`

(d) En prenant le gradient de `img2bis`

(e) En prenant la moyenne des deux

(f) En prenant celui qui a la plus grande norme

FIGURE 2.8 – Quel gradient choisir ?

5×10^5 pixels chacune. Ainsi, la zone `3` sur laquelle nous avions un grand nombre de calculs à faire contient environ 10^5 pixels. Et, `Matlab` refuse de créer des vecteurs si grands d'un seul coup (avec `zeros` par exemple). Dans un premier temps, nous avions utilisé des vecteurs de taille variable, ce qui augmente considérablement les temps de calcul. Comme `Matlab` acceptait de stocker des vecteurs si grands, nous avons ensuite eu l'idée de créer plusieurs vecteurs assez grands avec la commande `zeros` et de les concaténer. De cette façon, on fixe beaucoup plus vite la taille des vecteurs que l'on utilise.

Cette méthode était suffisante pour le cas du paragraphe 2.2.2. En ce qui

concerne le paragraphe 2.2.3, pour utiliser le « discrete Poisson solver », on doit résoudre un système du type $SF = B$ d'inconnue F, où B est un vecteur de taille N le nombre de pixels dans la zone 3, et S est une matrice de taille $N \times N$. Nous n'avons pas eu de problème pour créer le vecteur B, mais pour la matrice S : `Matlab` nous a très vite fait savoir qu'il avait atteint les limites de sa mémoire en renvoyant `Out of memory`. Heureusement, la matrice S est une matrice creuse [3] (i.e. contient un grand nombre de 0), nous avons donc pu utiliser la fonction `sparse` pour la créer. Pour résoudre le système $SF = B$, nous avons utilisé la fonction `bicgstabl` adaptée aux matrices creuses.

Un problème demeure, comme nous avons été contraints d'utiliser des boucles `for`, nos programmes en `Matlab` sont très lents. Au début, pour des images d'environ 10^5 pixels, on devait attendre plus d'une demi heure pour obtenir un panorama. Après optimisation, notamment en utilisant `sparse` au lieu de coder soi-même ce type de matrices, le temps de calcul a été réduit à un quart d'heure, ce qui reste trop lent à notre goût.

3. sparse en anglais

Conclusion

Mettre en œuvre des méthodes décrites dans des articles n'est pas aussi simples qu'il n'y paraît. Nous avons été confrontés à un grand nombre de difficultés. Néanmoins, nous sommes arrivés à des résultats très satisfaisants.

Pour améliorer le temps de calcul, nous pourrions récrire tous nos programmes en `C` ; nous ne serions d'ailleurs plus confrontés aux limites mémoire de `Matlab`.

En outre, nous aurions pu essayer d'autres méthodes pour régler les problèmes de continuité et de flou dans la zone `3`, comme celle d'alignement d'histogrammes.

FIGURE 1 – Un joli panorama !

Bibliographie

[1] Brown Matthew and Lowe David G. Recognising panoramas. Computer Vision, 2 :1218, 2003.

[2] Brown Matthew and Lowe David G. Automatic panoramic image stitching using invariant features. International Journal of Computer Vision, 74 :59–73, 2007.

[3] Hartley Richard and Zisserman Andrew. Multiple View Geometry in Computer Vision Second Edition. Cambridge University Press, 2003.

[4] Lowe David G. Object recognition from local scale-invariant features. Computer Vision, 2 :1150–1557, 1999.

[5] A. Vedaldi and B. Fulkerson. VLFeat : An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[6] Wiggin Edward. Ransac algorithm with example of finding homography. `http://www.mathworks.com/matlabcentral/fileexchange/30809-ransac-algorithm-with-example-of-finding-homography`, mar 2011.

[7] Martin A. Fischler and Robert C. Bolles. Random sample consensus : A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6) :381–395, 1981.

[8] Pérez Patrick, Gangnet Michel, and Blake Andrew. Poisson image editing. ACM Trans. Graph., 22(3) :313–318, jul 2003.