



RAPPORT DE PROJET IMAGE - HAI809I/HAI804I

Génération d'une image mosaïque

Réalisé par
Ange CLEMENT
Erwan REINDERS
Benjamin PRE

Sous la direction de
William PUECH
Bianca JANSEN VAN RENSBURG
Olivier STRAUSS

Année universitaire 2021-2022

Sommaire

Introduction	0
Avant propos	1
Techniques de génération d'images mosaïques	2
Modèle de base	2
Moyenne	3
Distance	4
Egalisation et spécification	5
Quadtree	7
Système de base de données	10
Détection et corrélation de points d'intérêts appliquées aux images mosaïques	11
Détection de points d'intérêts dans l'image	11
Représentation d'une image : Gradient et Hessien	12
Harris-Stephens	18
SIFT	20
Choix du descripteur	24
Correspondance entre deux images	25
Limites de la technique	28
Conclusion	30

Introduction

L'analyse et le traitement d'image ont connu de grandes avancées depuis les premières études sur le sujet au début du siècle dernier. D'abord utilisés pour des applications purement pratiques de compression et transmission de signaux numériques, ces procédés se sont ensuite étendus à de nombreux autres domaines, pour finalement se démocratiser complètement et faire partie intégrante de bon nombres d'appareils informatiques actuels (en témoigne les filtres de photos sur la plupart des applications modernes).

Rendu en grande partie possible par l'évolution des ordinateurs et de leur vitesse de calculs, le traitement d'image informatique moderne permet la manipulation d'un grand nombre de données, rendant accessibles ses applications dans des domaines plus liés à l'esthétique et la génération d'images, comme la réalisation d'image Mosaïques.

C'est dans ce contexte que notre travail prend forme. En effet, nous allons, au travers de ce rapport, dresser un tour d'horizon de quelques possibles méthodes de génération d'images Mosaïques que nous avons pu tester. Avant toute chose, une telle image peut se définir comme l'image générée à partir d'une grande base d'images de petites tailles appelées imagettes. Ce procédé reprend l'antique concept de génération de mosaïques décoratives par fragments colorés, en remplaçant ces derniers par des images numériques.

Nous allons donc dans un premier temps vous parler de techniques de génération d'une grande image Mosaïques basées sur des grilles plus ou moins régulières et des métriques de mise en corrélation différentes.

Ensuite, nous aborderons des techniques de mise en correspondance de pixels ou groupes de pixels par détection de points d'intérêts et leurs possibles applications à un tel projet.

Avant propos

Ce projet rentre dans le cadre d'un projet de manipulation d'images numériques. Il y sera souvent question de notions inhérentes à ce domaine d'étude qu'il peut être bon de rappeler brièvement. Ainsi, une image numérique telle que décrite durant ce rapport est une discrétilisation d'un signal continu de lumière faite par un capteur spécifique (caméra, appareil photo...) qui sera échantillonné sur une plage de valeurs dépendant par exemple du format de stockage du fichier.

Ainsi, une image numérique classique (telle que nous manipulons) peut se percevoir comme une matrice en deux dimensions de pixels. On peut ainsi, en renseignant la colonne ou la ligne de ce pixel, en récupérer sa valeur d'intensité lumineuse par exemple (ou encore sa valeur de chrominance, voir de teinte/saturation/lumière pour certains formats).

Nous travaillerons principalement sur des images en niveau de gris, ce qui signifie donc des images possédant un seul canal de valeur pour chaque pixel (256 niveaux de gris), ou une seule valeur d'intensité, contrairement au format RGB couleur qui possède trois valeurs de chrominances pour chaque pixel (3 canaux de 256 valeurs possibles chacun).

Les images étant définies comme des matrices de valeurs, il est possible d'opérer des calculs usuels d'algèbre linéaire sur ces valeurs, que nous verrons par la suite.

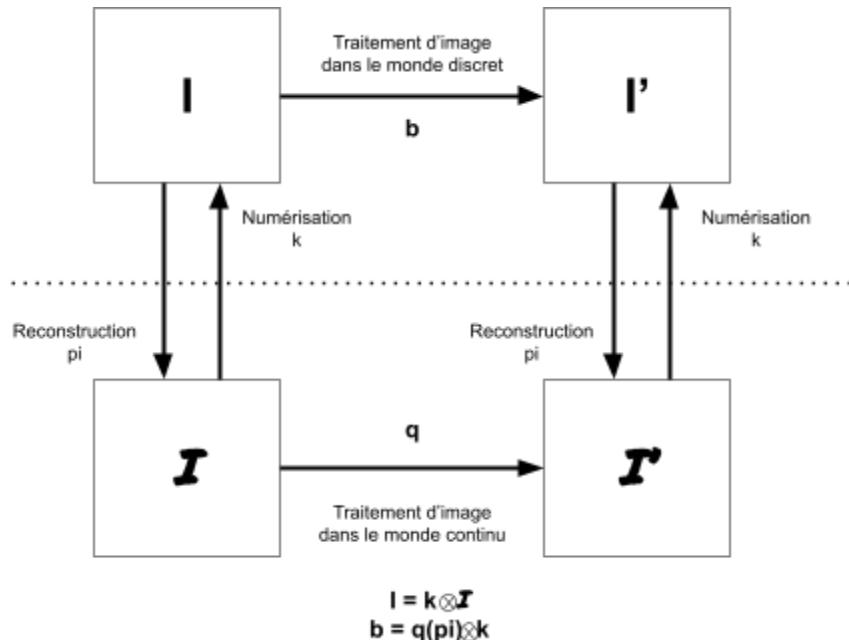


Schéma du traitement d'image en continu ou en discret

Ce schéma nous montre les liens entre monde continu et monde discret et numérique, et plus généralement l'intérêt que l'on pourrait avoir à définir une opération de traitement d'image dans un monde de représentation plutôt qu'un autre. On peut passer du monde continu au monde discret par échantillonnage (apportant son lot d'imprécisions), et inversement par reconstruction (avec ou sans interpolation).

Concernant le choix des images, nous sommes partis sur des images en niveaux de gris. Cela peut paraître étrange de se limiter en format d'image, étant donné que la génération d'images mosaïque dans notre cas a un intérêt principalement esthétique, mais cela nous permet par la suite de mettre en place des techniques de détections de primitives géométriques plus poussées.

Techniques de génération d'images mosaïques

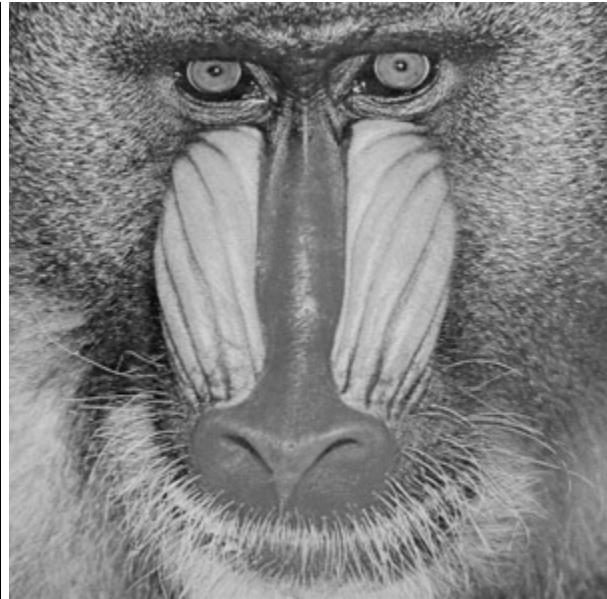
Vous l'aurez compris, le but de ce projet consiste à expérimenter les différentes techniques de génération d'une grande image mosaïque à partir d'une grande base d'imagettes. Le principe de la méthode consiste à prendre une grande image, la découper en blocs de petites tailles, puis de remplacer chaque bloc de la grande image par l'imagette la plus similaire issue de la grande base d'imagettes.

Modèle de base

On définit les variables pour mettre en place un système de comparaison d'imagettes. Premièrement, on découpe l'image d'origine avec un partitionnement en grille régulière. On se retrouve avec un ensemble de blocs qui vont recevoir une imagette. Ensuite, on réduit la taille des imagettes pour qu'elles correspondent avec les blocs. Ainsi, on peut effectuer une comparaison entre les imagettes et le bloc.

I : image d'origine
 O : image de sortie
 S : ensemble d'imagettes

```
Algo Mosaique(I, O, S) :
B <- Decouper_en_bloc(I)
Pour chaque b dans B :
  i <- min(s dans S, score(s, b))
  dessiner(i, O)
Fin pour
fin
```

*Liste imagettes**Image originale*

Moyenne

La première stratégie est de choisir une imagette en fonction de sa moyenne. On recherche donc l'imagette ayant la moyenne la plus proche de celle du bloc courant.

s : imagette
 b : bloc de l'image d'origine

```
Algo score(s, b):
  Return abs(moy(b) - moy(s))
Fin
```

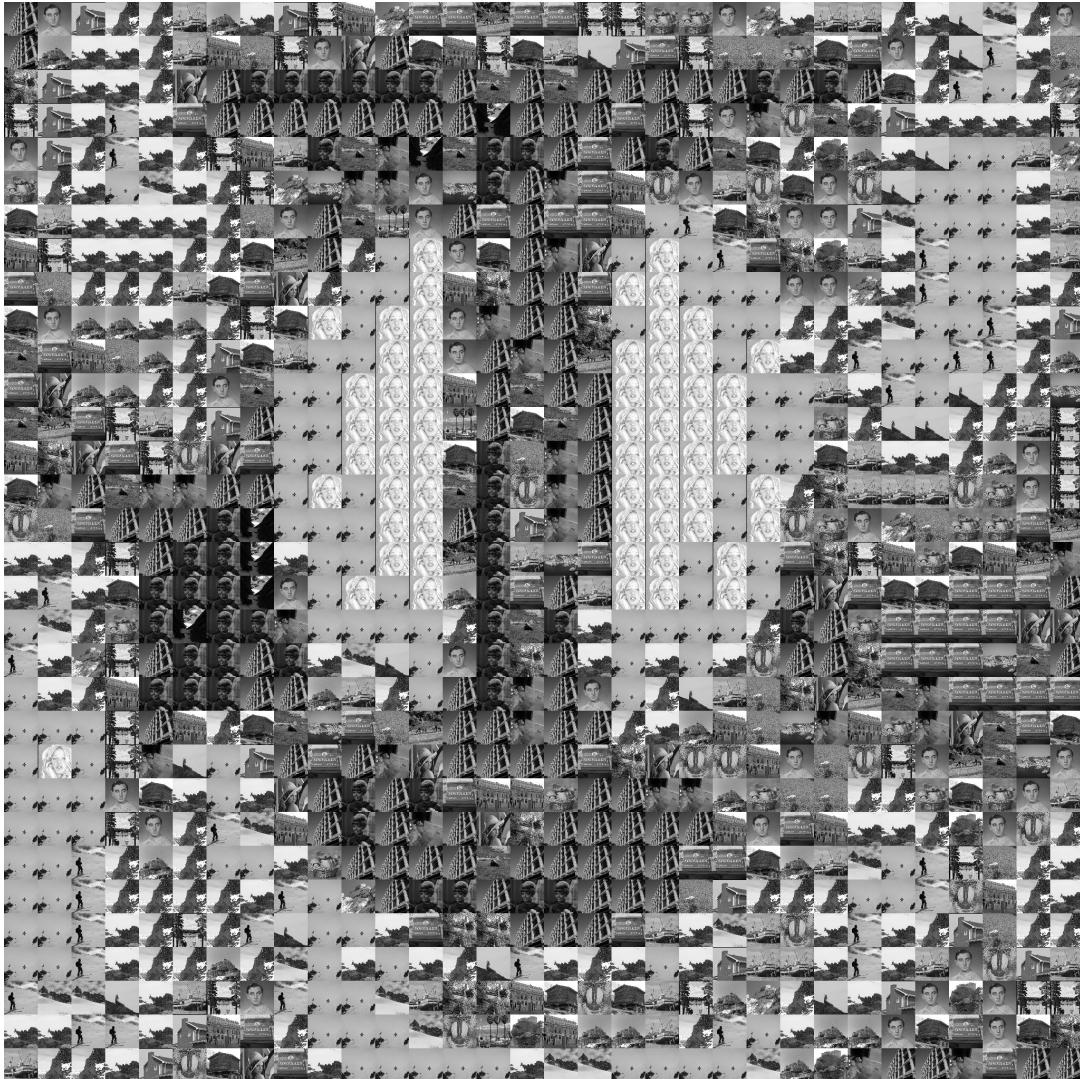


Image mosaïque par moyenne

Distance

La moyenne permet d'obtenir une bonne couleur d'un bloc, mais ne fait pas la différence entre une image moitié-claire/moitié-sombre et une image uniforme ayant la même moyenne. Afin de prendre en compte cela, on doit choisir une autre métrique. On calcule la distance entre le bloc et les imagettes. Le calcul de distance choisi est "l'erreur quadratique moyenne" (somme des différences au carré, divisé par le nombre d'éléments). Comme il s'agit d'une comparaison entre des imagettes ayant toutes la même taille, il est inutile de diviser par le nombre d'éléments.

Algo score(s, b):

$S \leftarrow 0$

Pour chaque pixels de s:

 Pour chaque pixels de b:

```

S <- S + pow(s(i, j) - b(i, j), 2)
Fin pour
Fin pour
Return S
Fin

```

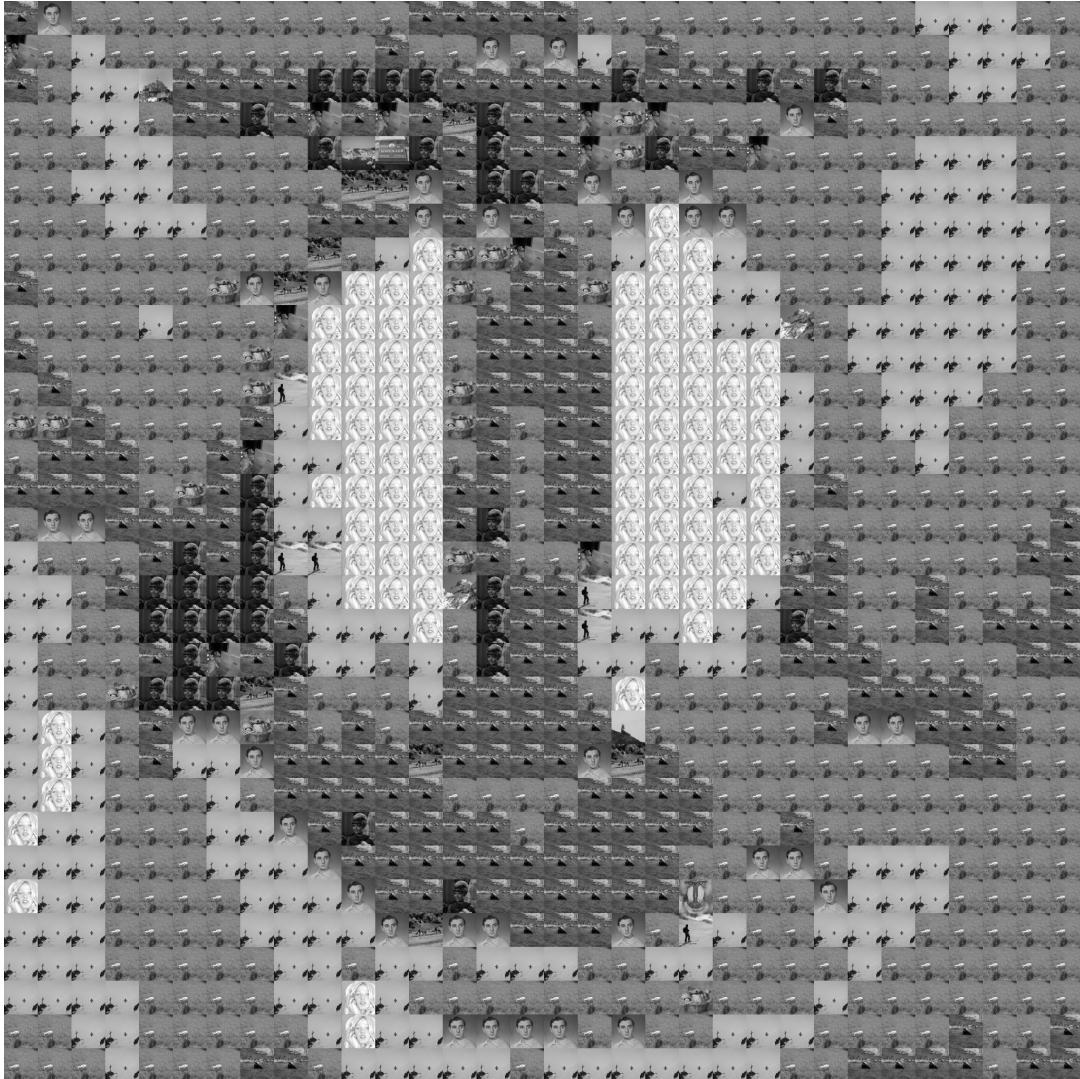


Image mosaïque par distance

Egalisation et spécification

L'image mosaïque par distance préfère utiliser les imagettes proches des blocs, mais comme les blocs sont des petites parties de l'image, ils ont une couleur presque uniforme (les pixels proches sont corrélés). De plus, on garde les couleurs exactes des imagettes.

On s'autorise à changer les couleurs des imagettes. On peut donc trouver une nouvelle technique qui consiste à tout d'abord égaliser les imagettes et le bloc, puis de calculer la

distance de l'imagette afin de choisir la meilleure. Ensuite, on applique l'égalisation inverse du bloc sur l'imagette choisie avant de l'appliquer.

```
Algo score(s, b):
  s' <- egalisation(s)
  b' <- egalisation(b)
  Return distance(s', b')
Fin
```

```
Algo dessiner(i, O) :
  i' <- egalisation_inverse(b, i)
  appliquer(i', O)
Fin
```

Ainsi, on ne choisit les imagettes que sur la différence entre les couleurs dans l'imagette, et on applique cette imagette en utilisant des couleurs proches de celles du bloc.

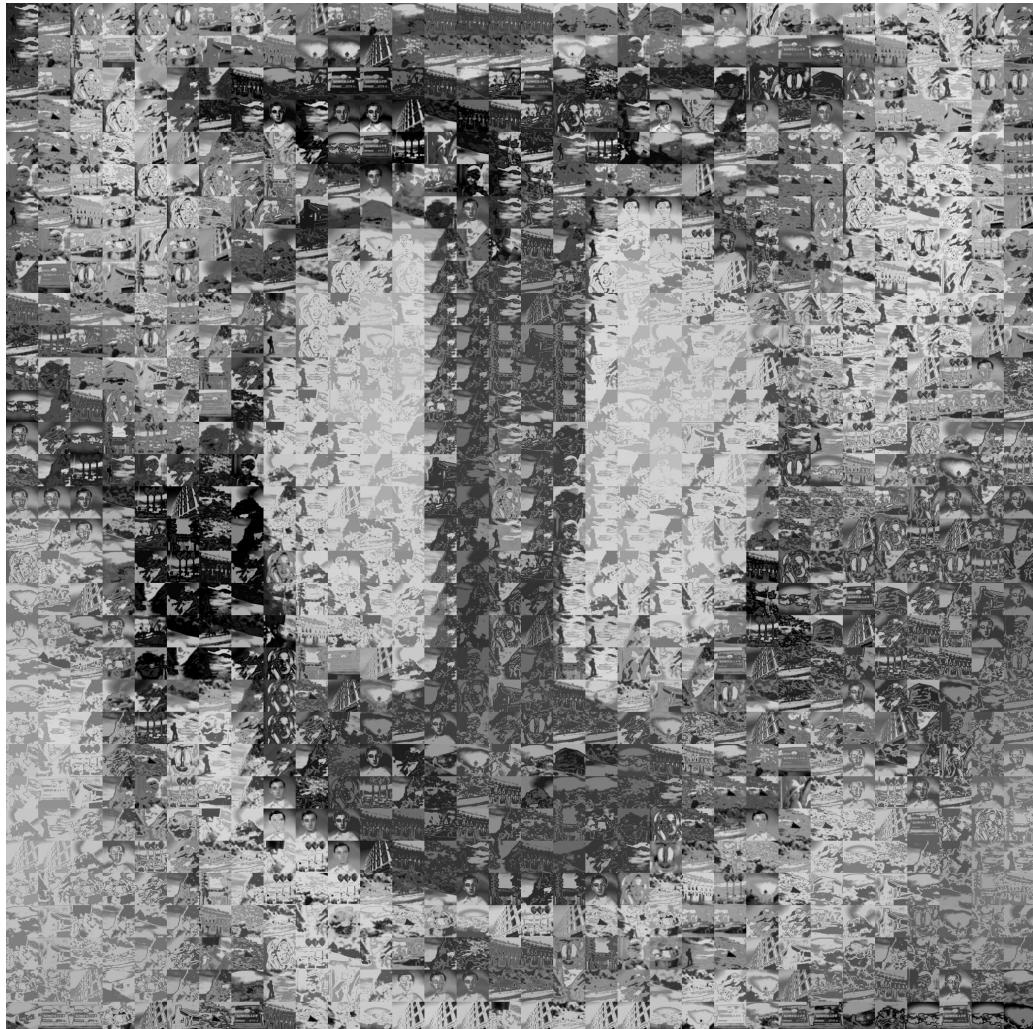
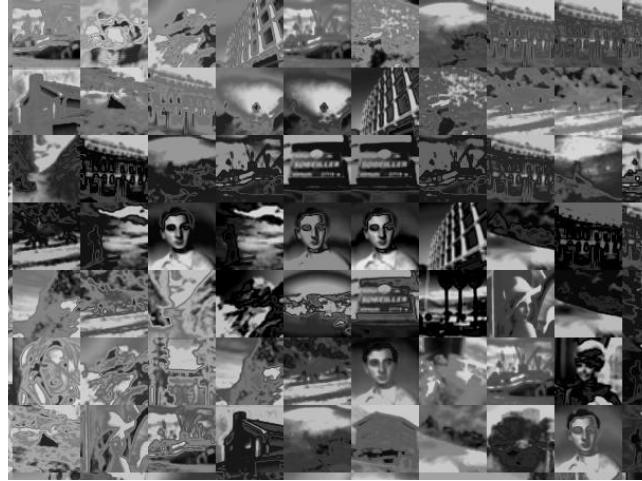


Image mosaïque par égalisation et spécification



Zoom sur l'œil droit. On peut déterminer que la zone autour de l'œil en noir est bel est bien composé avec les imagettes.

Quadtree

Jusque-là, on a toujours partitionné l'image d'origine en grille régulière. Cependant, il existe d'autres techniques de découpage. Comme on possède un score sur le bloc, on peut mettre en place un quadtree basé sur ce score.

```
I : image d'origine
O : image de sortie
S : ensemble d'imagettes
Seuil : seuil pour le quadtree
TailleMinimum : facteur d'arrêt pour le quadtree
```

```
Algo Mosaique(I, O, S) :
B <- Decouper_en_quatre(I)
Pour chaque b dans B :
    i <- min(s dans S, score(s, b))
    Si score(i) > Seuil et taille(i) > TailleMinimum :
        Mosaique(b, O, S)
    Sinon
        dessiner(i, O)
    Fin si
Fin pour
Fin
```



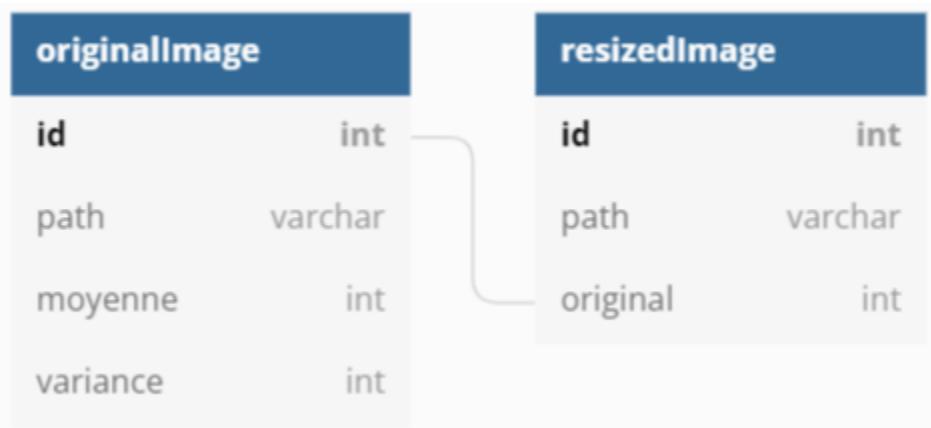
Image mosaïque avec quadtree par égalisation et spécification



Zoom sur l'arête du nez. L'image du skieur permet d'obtenir un dégradé horizontal de gauche à droite qui possède un score suffisamment bon pour être placé. De même, la montagne permet d'obtenir un dégradé horizontal de droite à gauche.

Système de base de données

Bien que non implémenté dans la version finale du projet, nous avons fait et commencé à implémenter un système de base de données afin de gérer les imagettes de la mosaïque, avec le schéma de BDD suivant :



La base de données a été réalisée avec SQLite. SQLite est une librairie C permettant l'implémentation d'une base de données de façon rapide et légère tout en assurant toutes les fonctionnalités basiques de SQL. SQLite ne permet pas de créer des fonctionnalités avancées de SQL, comme de triggers, mais ce genre de fonctionnalités ne nous est pas utile dans le cadre de ce projet. En utilisant SQLite, nous voulions créer une base de données contenant tous les pré calculs, afin de pouvoir préfiltrer les imagettes et réduire le temps de calcul, à la fois pour les pré calculs lourds (mipmapping) et pour réduire le nombre d'images.

Finalement, nous stockons un ensemble de pré calculs en amont et en local, afin de ne pas tout devoir recalculer ensuite, comme la moyenne ou encore les points d'intérêts que l'on va aborder ensuite.

Détection et corrélation de points d'intérêts appliquées aux images mosaïques

Les points d'intérêts peuvent se décrire comme des points possédants des caractéristiques supplémentaires, les distinguant ainsi des autres points de l'image. Ces points d'intérêts, pour une image donnée, peuvent être mis en corrélation ensuite avec d'autres points d'intérêts d'une autre image, et ainsi obtenir une sorte de score de ressemblance entre ces deux images, basés sur la distance entre leurs points d'intérêts respectifs.

Détection de points d'intérêts dans l'image

Ce sont historiquement les coins d'un objet d'une image que l'on a d'abord cherché à définir, mais la plupart des détecteurs de points d'intérêts ne trouvaient pas seulement des coins, mais également tout un tas d'autres points plus ou moins pertinents dans le traitement numérique ensuite. Pour pallier ce problème, il est possible d'opérer des phases de filtration de ces points.

Toutes ces méthodes ont donc pour but de donner en premier lieu une valeur d'intérêt à un point donné de l'image à analyser. Elles n'opèrent pas toutes de la même manière pour arriver à un résultat pertinent pour l'analyse ensuite. Historiquement, ont été mis en place en premières des méthodes de détection de points d'intérêts par détection de contours dans une image, les contours servant ensuite à repérer les coins d'un objet. Par la suite, des méthodes basées sur l'intensité (détection de fortes variations) ont vu le jour, comme le détecteur de Harris ou encore la méthode SIFT. Ont également vu le jour des méthodes plus complexes basées cette fois-ci sur des modèles paramétriques par déformation d'un modèle paramétrique d'un coin, pour le faire à terme se rapprocher les plus possibles en niveau de gris des valeurs de voisinage d'un autre coin. Dans ce rapport, nous aborderons principalement des méthodes de détection de points d'intérêts basées sur l'intensité et la variation locale d'intensité d'un pixel d'une image.

Ces méthodes peuvent être employées dans divers domaines, comme la cartographie, la détection d'objets, l'assemblage de photographies, la détection de contenu dans une image, mais également la recherche d'images par contenu. C'est ce dernier aspect qui nous a poussés à entrevoir une utilisation de ces techniques pour de la construction d'images mosaïques.

En effet, une fois notre image originale partitionnée en un ensemble de zones dépendant de l'aspect de la grille de départ, il est possible de venir calculer les points d'intérêts de ces zones, pour en extraire des primitives géométriques. Ensuite, il devient possible de comparer toutes ces primitives géométriques avec celles précédemment calculées pour chacune de nos imagettes, et de convenir de la meilleure imagette à prendre pour cette zone de l'image originale.

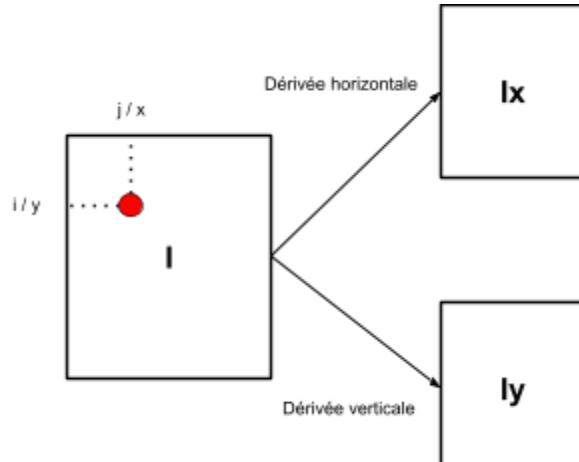
Représentation d'une image : Gradient et Hessien

Avant de commencer à mettre en correspondance les composantes géométriques d'une image avec celles d'une autre, il faut dans un premier temps les détecter au sein de ces images. Plusieurs techniques utilisant le paradigme de variation d'intensité sont possibles à implémenter. Elles utilisent souvent plusieurs représentations différentes d'une image source, représentations obtenues par des traitements précis de cette image. Plus précisément, le traitement qui nous intéresse ici est le calcul du Gradient d'une image (vertical ou horizontal) et le calcul de la matrice Hessienne H.



Image initiale (ndg)

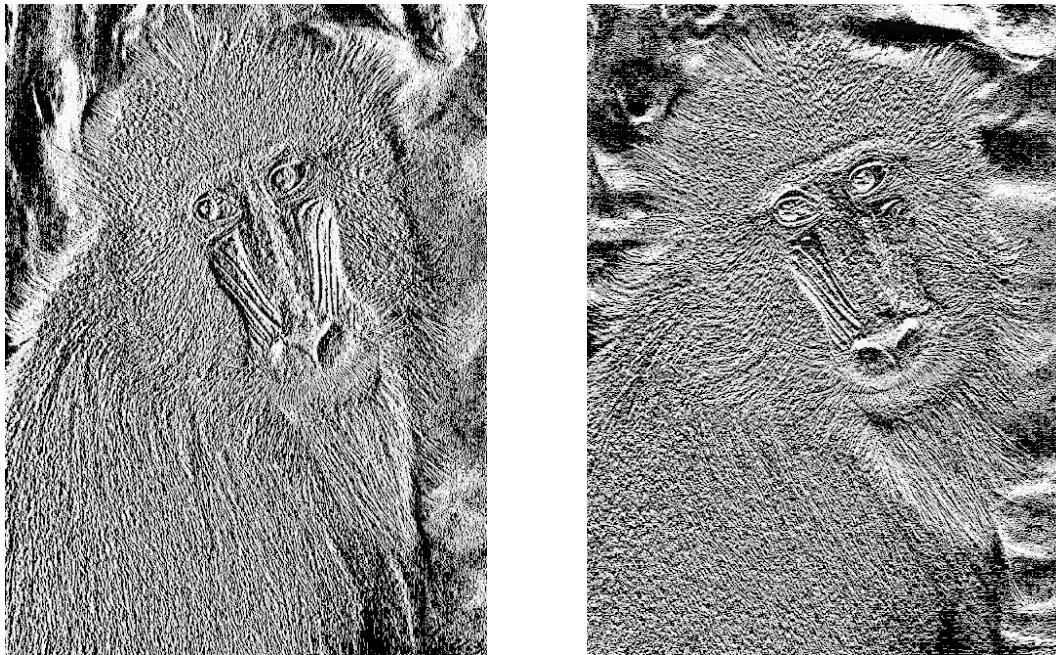
Le Gradient peut être vu comme la dérivation d'une matrice image, dans une direction verticale ou horizontale. Une dérivation peut se traduire comme une variation locale d'une quantité monodimensionnelle ou bidimensionnelle. Appliqué à une image en deux dimensions, cela nous donne une bonne première description de l'image :

Schéma du gradient de l'image initiale I

Pour calculer cette dérivée verticalement et horizontalement, on va passer par un formalisme connu en monde discret qui se base sur les formules suivantes :

$$Ix(x, y) = \frac{\partial I(x, y)}{\partial x} = \frac{\Delta I(i, j)}{\Delta j} = I(i + 1, j) - I(i, j)$$

$$Iy(x, y) = \frac{\partial I(x, y)}{\partial y} = \frac{\Delta I(i, j)}{\Delta i} = I(i, j + 1) - I(i, j)$$

Images dérivées en x (gauche) et y (droite) de notre image initiale

Cette première dérivée de notre image nous permet de définir la direction globale de la plus grande pente en un point donné de notre image (vecteur 2D). Le Gradient étant une description 2D d'un point d'une image, il est possible d'en calculer sa norme et de l'inclure à une troisième image :

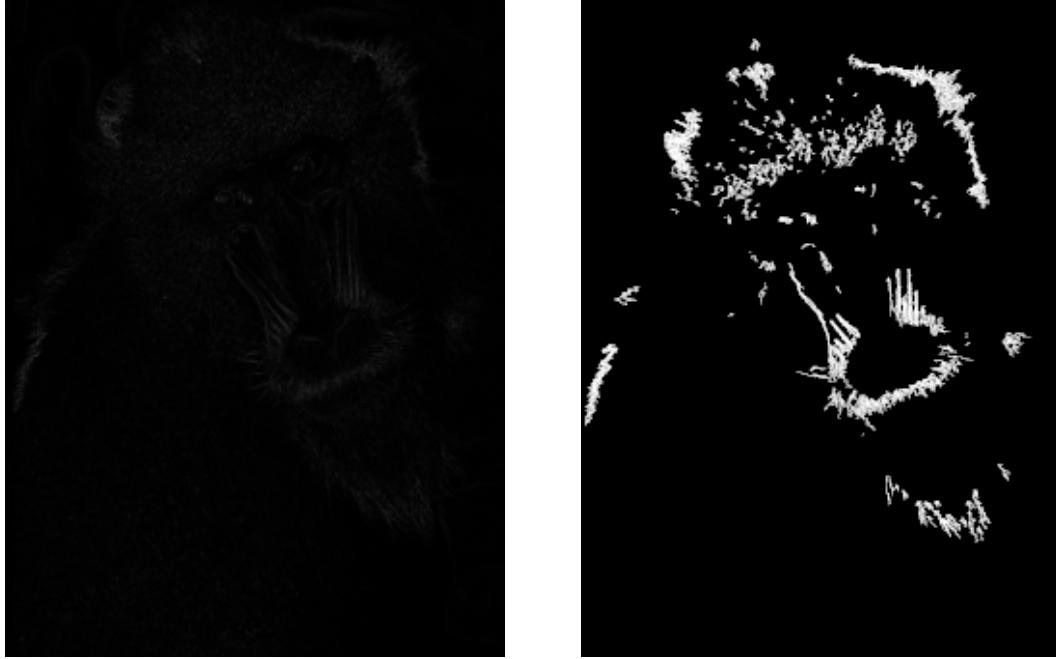


Image de la norme du gradient appliquée à notre image (seuillée par hystérisis à droite)

Un seuillage basique de la norme du gradient nous donne les extrema locaux du Gradient, mais cela ne suffit pas pour détecter des zones précises d'une image, comme des contours, car on peut avoir des contours n'ayant pas la même intensité dans notre image. Ce type de points de contours dans notre image répond à un critère supplémentaire : il doit également être un maximum local par rapport à la norme du Gradient, et cela, dans la direction du Gradient (car le Gradient est la direction de la plus grande pente).

Cependant, nous ne voulons pas simplement détecter les contours d'un objet de l'image, mais les points d'intérêt de celle-ci. La différence majeure entre ces deux notions réside dans le fait qu'un point d'intérêt à de fortes chances d'être un point de contour (angle), et qu'un point de contour n'est pas nécessairement un bon point d'intérêt (n'apporte pas d'informations géométriques en plus). Par la suite, on verra que certaines méthodes vont filtrer les points d'intérêt faisant partie des contours.

Un point d'intérêt de notre image peut alors s'exprimer comme un point dans lequel le gradient va varier dans toutes les directions (ici en x et en y) :

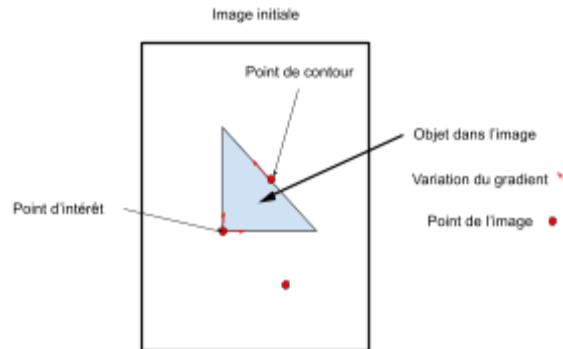


Schéma explicatif d'un point d'intérêt par le gradient

Ainsi, pour obtenir ces points de forte variation de la dérivée, on va appliquer une seconde dérivée, à ces deux images de Gradient afin d'obtenir une dérivée du second ordre de ces points d'intérêts.

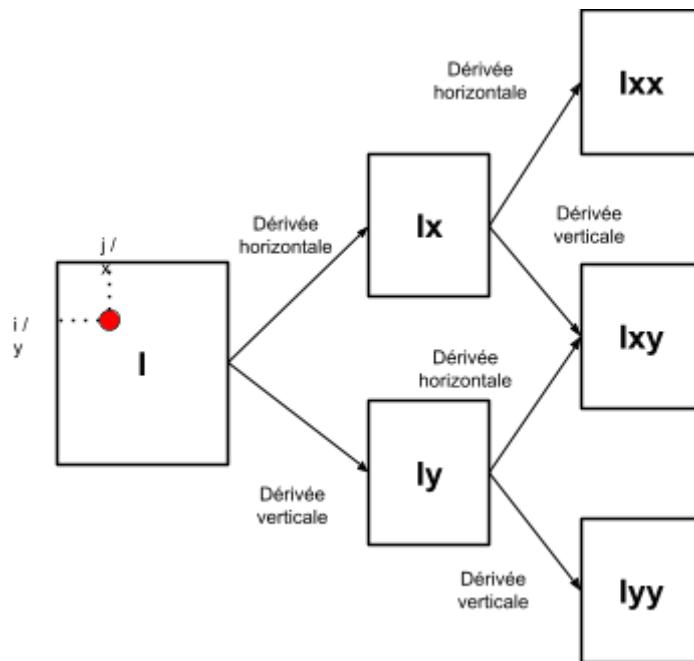


Schéma de la dérivée seconde de l'image initiale I

Cette dérivée seconde nous donne une nouvelle matrice, la matrice Hессienne :

$$H(x,y) = \begin{pmatrix} I_{xx}(x,y) & I_{xy}(x,y) \\ I_{yx}(x,y) & I_{yy}(x,y) \end{pmatrix}$$

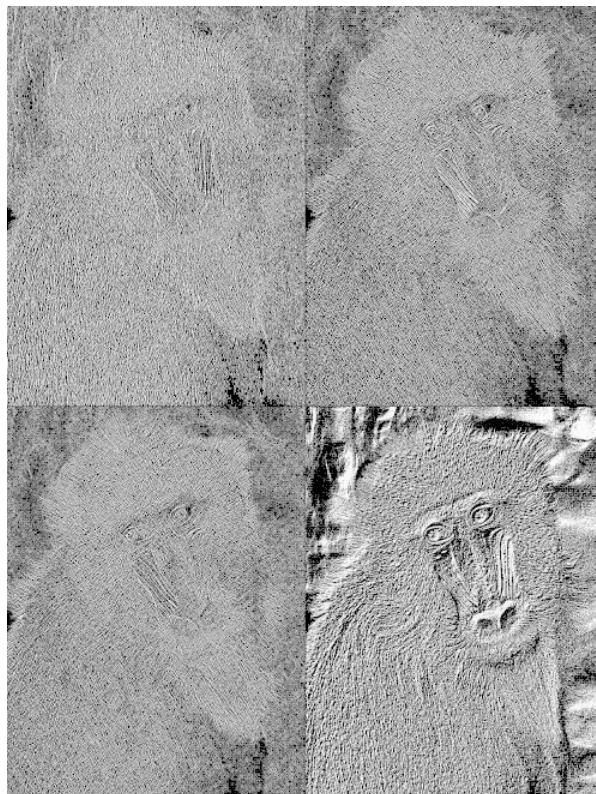


Image de la matrice Hessienne de notre image initiale

C'est en combinant ces diverses informations sur notre image que l'on va pouvoir ensuite l'analyser géométriquement.

C'est ce qui est implémenté dans l'algorithme de Rosenfeld :

$$KR = \frac{I_{xx}I_y^2 + I_{yy}I_x^2 - 2I_{xy}I_xI_y}{I_x^2I_y^2}$$



Image des points d'intérêts obtenus par Rosenfeld

Un autre algo décrit par Beudet, se base également sur les matrices de dérivé première et seconde de notre image (déterminant du Hessian) :

$$KR = I_{xx}I_{yy} - I_{xy}^2$$



Image des points d'intérêts obtenus par Beaudet (seuil à droite)

On remarque beaucoup de points d'intérêts, voire trop en réalité. Cela est dû au fait que l'on manipule des dérivées qui sont par nature instables, et des dérivées de dérivées instables, elles-mêmes instables. Cela va générer beaucoup de faux positifs dans nos résultats.

Il nous faut donc un algorithme de détection ne se basant plus cette fois-ci sur des dérivées d'images trop instables. C'est le cas des deux autres algorithmes que nous allons présenter ensuite.

Harris-Stephens

La méthode des points fixes d'Harris-Stephens va se baser sur une approximation de la matrice Hessienne par un filtrage Gaussien pour opérer des détections de points fixes. En posant M notre matrice du faux Hessien, la formule de l'algorithme peut se traduire ainsi :

$$KH = \det(M) - k * \text{trace}(M)$$

On fait généralement varier le k entre 0.04 et 0.15.



Image des points d'intérêts obtenus par Harris-Stephens G(5,0.1) k=0.09



Image des points d'intérêts obtenus par Harris-Stephens G(5,0.1) k=0.04



Image des points d'intérêts obtenus par Harris-Stephens G(1,0.5) k=0.04

Le résultat est mieux que dans les deux méthodes précédentes, avec toujours beaucoup de points d'intérêts. Cependant, ces points de contour et pas d'intérêt semblent plus facilement filtrables que ceux obtenus précédemment.

SIFT

Il s'agit d'une méthode mise en place dans les années 1999 par un chercheur canadien David G Lowe, permettant d'extraire des points d'intérêts (phase de détection), puis d'apporter une description géométrique sur ceux-ci (phase de description). Entre ces deux phases, ces points d'intérêts sont filtrés pour récupérer que les plus pertinents pour notre traitement (filtrage par valeur de contraste sur la valeur du Laplacien obtenu par DoG, filtrage des points de contours par seuil sur le rapport de courbure principale via une approximation de la matrice Hessianne).

Son principe réside en grande partie dans la localisation de points d'intérêts par un vecteur descripteur que l'on va comparer aux caractéristiques des points trouvés dans une base de données, via différentes fonctions de mise en correspondance.



Image des points d'intérêts obtenus par SIFT (3080)

Dans cette représentation, les points d'intérêts sont au centre des cercles, de rayon un certain facteur d'échelle σ , et de centre xy . Ce facteur d'échelle est plus important pour des points d'intérêts trouvés à des résolutions plus petites de l'image. On détermine pour ces points également une orientation intrinsèque servant de base à la construction d'un histogramme des orientations locales des contours. Cet histogramme est seuillé, normalisé (128 dimensions), et va ensuite composer le fameux descripteur SIFT du point d'intérêt. Tous ces descripteurs vont

ensuite former l'équivalent d'une réelle description géométrique complète de l'image (matrice descriptive du jeu de données de l'image).

Une fois les points d'intérêt déterminés, s'ensuit une étape de convergence et de filtrage de ces points d'intérêt (éliminer les moins pertinents et améliorer leur localisation). L'orientation intrinsèque ne dépend que du contenu local de l'image autour du point d'intérêt (rayon facteur d'échelle considéré).

On va détecter les points dans l'espace des échelles à trois dimensions : (x,y,σ)

Cette représentation nous permet de calculer le facteur de gradient d'échelle σ comme le résultat de la convolution d'une image I par un filtre Gaussien de paramètre σ :

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

En traitement d'image numérique, la convolution d'une image par un filtre est l'application des coefficients de la matrice du filtre en tout point de l'image de départ. En appliquant ce traitement de convolution à l'image originale, on va en quelque sorte diminuer en intensité les détails de rayons inférieurs à σ . Ainsi, pour étudier les éléments de taille σ , on va construire une image appelée différence de gaussienne (DOG) :

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

k : paramètre fixe de l'algorithme dépendant de la finesse de discréétisation de l'espace des échelles voulues.

Convoluer une image par une Gaussienne revient à flouter en quelque sorte cette image par une fonction Gaussienne, dont l'intensité dépendra essentiellement du facteur σ (plus σ est fort, et plus le flou est intense car écart à la moyenne centrée sur le pixel d'application étendu). Les contours d'une image étant des zones d'intensité marquée et le flou permettant d'atténuer le bruit dans une image, faire la différence d'une image initiale avec son image floutée va nous donner le bruit de cette image, donc les zones de forte variation, donc les contours.

De la même manière que pour la matrice Hessienne, on peut donc déterminer un point d'intérêt comme un point où un extremum du DoG est atteint :

$$D(x + \delta x, y + \delta y, s\sigma) \text{ avec } \delta x \in \{-1, 0, 1\}, \delta y \in \{-1, 0, 1\}, s \in \{k(-1), 1, k\}$$

On réalise ceci en construisant une pyramide d'images où on va venir progressivement réduire la résolution de l'image de départ, par deux souvent, multipliant le facteur d'échelle par deux. Ceci préfigure la première étape de l'algorithme.

Dans ce même algorithme, k est calculé de telle sorte qu'au final l'espace discrétré des facteurs d'échelles considérés soit le reflet d'une progression géométrique :

$$(pour chaque changement d'octave) k^p \sigma_0 = 2^t \sigma_0$$

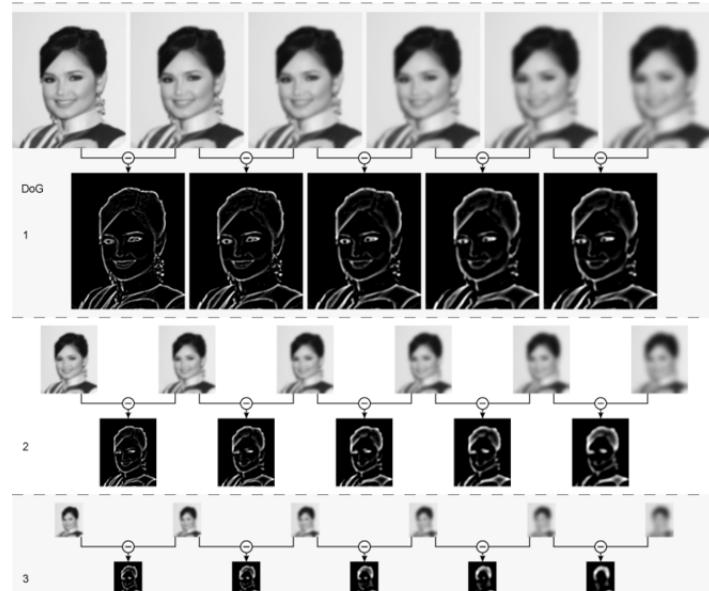


Image de la pyramide pour une image donnée (en haut à gauche)

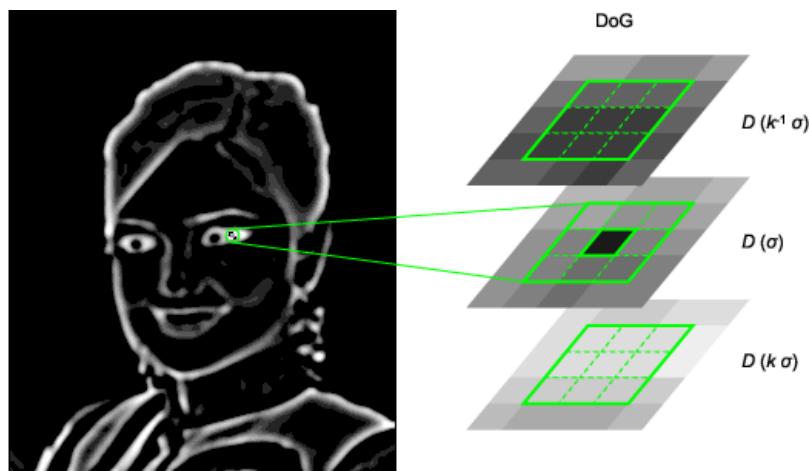


Image du calcul de la valeur d'un point de l'image dans la détermination des points d'intérêts

Avant d'en arriver au même résultat que la première image montrée, il faut opérer, suite à cette phase de construction des points d'intérêts par pyramide de DoG, une phase de filtration. En effet, on va détecter beaucoup de points d'intérêts, notamment dans les images de faible résolution au sein de la pyramide. Comme dit plus haut, pour pallier ce problème, on va opérer une étape de reconvergence des positions de ces points, puis une autre étape

d'élimination des points de faible contraste et des points de contours (car respectivement trop proche d'être du bruit ou pas intéressants géométriquement parlant).

Dans un premier temps, on peut recadrer les points des maximums locaux trouvés par développement de Taylor à l'ordre deux de la fonction du DoG :

$$D(x,y,\sigma)^T = D + D(x, y, \sigma)t = D + \frac{\partial D t}{\partial(x,y,\sigma)t} + \frac{1}{2} \partial(x, y, \sigma)t \frac{\partial^2 D}{\partial(x,y,\sigma)t^2}(x, y, \sigma)t$$

En prenant le point annulateur de la dérivée de cette fonction en notre point d'intérêt courant comme position véritable de ce point d'intérêt, il nous reste alors à opérer une interpolation pour déterminer le déplacement que ce point d'intérêt devra effectuer afin d'être recadré. Pour éliminer les points de faible intensité, on peut ensuite opérer un seuillage classique.

Ensuite, on peut vouloir éliminer les points sur les arêtes. En effet, ces points ont la particularité de se voir attribuer de fortes valeurs par le DoG, favorisant les extrema locaux instables sensibles au bruit. Pour ce faire, on va analyser la courbure principale du point le long du contour sur lequel il s'inscrit grâce au Hessien, de la manière suivante :

$$\frac{\text{tr}(H)^2}{\det(H)} < \frac{(rth + 1)^2}{rth} \text{ où } rth \text{ est un seuil (=10 dans la méthode originale)}$$

Ainsi, si un point ne respecte pas cette contrainte, il est rejeté car localisé par l'algorithme sur une arête.

Par la suite, l'algorithme va, en plus d'attribuer une position et une valeur de rayon aux points d'intérêts, leur opérer une assignation de direction grâce à la matrice image résultant de la convolution avec un filtre moyenneur Gaussien : $L(x,y,\sigma)$.

Pour chaque voisin de notre point d'intérêt, on va calculer ces informations (norme et angle du Gradient) :

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \arctan\left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right)$$

L'algorithme va ensuite construire l'histogramme de 36 intervalles de 10 degrés chacune. Les pics sont donc ici les orientations dominantes. Ainsi, toutes les orientations

permettant d'atteindre au moins 80% de la valeur maximale sont prises en compte. On a donc un point d'intérêt qui sera alors défini par quatre paramètres :

$$(x, y, \sigma, \theta)$$

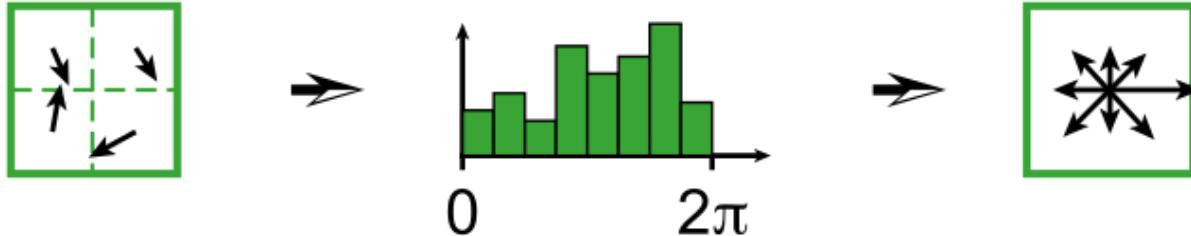


Schéma d'attribution de l'orientation

Enfin, on va au travers de cette étape assigner un vecteur descripteur à chacun des points d'intérêts via les informations précédentes. Pour ce faire, on commence par appliquer une rotation localement d'angle égal à l'orientation du point d'intérêt, mais de sens opposé. On considère ensuite, toujours autour du point d'intérêt, une zone de 16x16 px que l'on partitionne en 4x4 zones de 4x4px. On va calculer sur chaque zone le même histogramme d'orientation que précédemment, mais pour 8 plages d'intervalles.

On concatène ensuite ces 1- histogrammes à 8 intervalles puis on les normalise. Afin de réduire les problèmes de détection liés à des variations de luminosité, on peut définir un maximum sur cet histogramme à 0.2 et normaliser à nouveau. On obtient au final un descripteur SIFT de dimension 128 de notre point d'intérêt.

Choix du descripteur

On essaye ici de faire de la détection d'objets, passant par des notions de couleur, de forme ou encore de mouvement. Ainsi, avec la méthode SIFT, deux images différentes auront beaucoup de chance d'avoir un descripteur SIFT différent, et donc peu de points d'intérêts en corrélations. De même que deux images se ressemblent fortement à une transformation près (couleur ou géométrique), auront un descripteur SIFT proche. Cet algorithme est donc relativement robuste pour la détection et la correspondance géométrique au sein d'une image, en plus de proposer un réel critère discriminant. De plus, les descripteurs qu'il emploie sont invariants à l'orientation et à la résolution de l'image, et peu sensibles à son exposition, à sa netteté ainsi qu'au point de vue 3D.

Enfin, la mise en correspondance n'est que faiblement impactée par l'augmentation de la taille de la BDD.

Par ailleurs, la méthode SIFT produit une description plus complète des points d'intérêts, permettant un meilleur raffinement dans l'étape suivante de mise en correspondance.

Pour toutes ces raisons, nous avons finalement décidé de partir sur la méthode SIFT pour la détection de points d'intérêts, que nous implémenterons via la librairie OpenCV en C++.

Correspondance entre deux images

Pour la correspondance entre deux images, on va établir une table de mise en correspondance des descripteurs de la première image (image question) avec ceux de l'image référence, et ce par la distance euclidienne. On peut alors voir apparaître des clusters, dans lesquels la mise en correspondance est cohérente. On supprime au passage les clusters trop petits et les correspondances aberrantes. Toute ces correspondances et clusterings se basent sur des modèles probabilistes de manipulation d'un grand ensemble de jeu de données par transcription vectorielle.

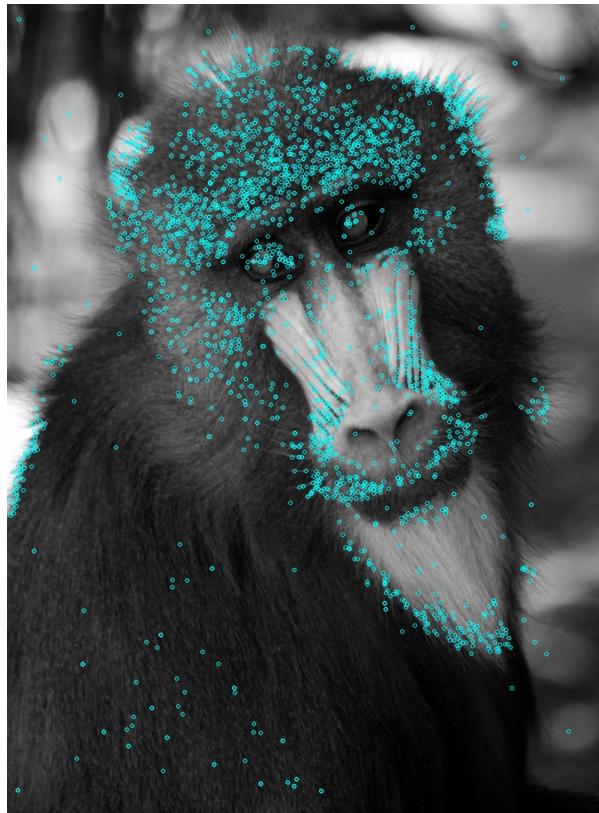


Image des points d'intérêts obtenus par SIFT : octaves 3, seuil contraste 0.04, seuil arrête 10, sigma 1.6

On peut appliquer plusieurs méthodes de correspondance différentes pour nos images, étant donnée la taille des descripteurs SIFT ($\text{nbPts} * 128$). Ces méthodes vont venir relier des points d'intérêts de deux images ayant la plus faible distance. Ainsi, pour chaque point d'intérêt d'un bloc que l'on cherche à remplacer par une imagette de l'image originale, on va venir établir

une table de correspondance avec tous les points d'intérêts de toutes les imagettes de notre base de données, et ainsi trouver l'imagette qui correspond le mieux géométriquement au bloc.

Exemple de code OpenCV :

Lecture des images dans une matrice

```
Mat img1_col = imread(nom1, IMREAD_ANYCOLOR | IMREAD_ANYDEPTH);
Mat img2_col = imread(nom2, IMREAD_ANYCOLOR | IMREAD_ANYDEPTH);
Mat img1 = imread(nom1, IMREAD_GRAYSCALE);
Mat img2 = imread(nom2, IMREAD_GRAYSCALE);
```

Création du détecteur de points d'intérêts (SIFT) et des liste de points fixes et matrices descripteurs

```
Ptr<SIFT> detector = SIFT::create();
std::vector<KeyPoint> keypoints1, keypoints2;
Mat descriptors1, descriptors2;
```

Détection des points d'intérêts et calcul des descripteurs SIFT

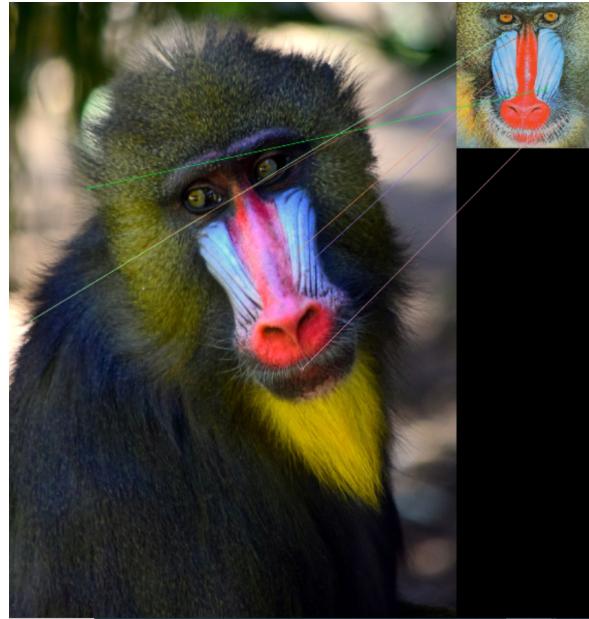
```
detector->detectAndCompute(img1, noArray(), keypoints1, descriptors1);
detector->detectAndCompute(img2, noArray(), keypoints2, descriptors2);
```

Mise en correspondance de chaque points d'intérêts avec les nb_match les plus proches

```
Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create(descriptorMatcherType);
std::vector< std::vector<DMatch> > knn_matches;
matcher->knnMatch(descriptors1, descriptors2, knn_matches, nb_match);
```

Filtration des distances par ratio (on ne veut que les meilleurs résultats de correspondance)

```
std::vector<DMatch> good_matches;
for (size_t i = 0; i < knn_matches.size(); i++){
    if (knn_matches[i][0].distance < ratio_thresh * knn_matches[i][1].distance){
        good_matches.push_back(knn_matches[i][0]);
    }
}
```



Mise en correspondance des images par la méthode BRUTE FORCE et KNN, k=2



Mise en correspondance des images par la méthode FLANN et KNN, k=2

La méthode FLANN disponible dans OpenCV est une version accélérée de l'algorithme de recherche de meilleurs voisins brute force classique (Fast Library for Approximate Nearest Neighbors), souvent utilisé avec des descripteurs de points fixes comme SIFT ou SURF, qui est une version accélérée de SIFT et plus robuste à certaines transformations d'images.

On prend un nombre de points d'intérêts à match à chaque autre point d'intérêt de deux minimums (knn avec $k = 2$), afin de pouvoir opérer la mise en comparaison ensuite des points par application du ratio de la distance, tel qu'expliqué dans le papier de Lowe (inventeur de la technique SIFT).

On remarque que les deux images prises en compte auraient tendance à voir plus de points d'intérêts mis en relation, mais ce n'est pas le cas dans les faits observables. Par ailleurs, les techniques de mise en correspondance fonctionnent mieux pour des images proches, à une transformation géométrique près, et pas deux individus différents.

Ainsi, on peut vouloir faire un algorithme combinant un système de mise en adéquation de points d'intérêts pour des blocks en possédant (dans l'image initiale), et de calculs de métriques autres (solution plus classique), pour des blocs vides de points.

Limites de la technique

Premièrement, la mise en correspondance des points d'intérêts de chacun des blocs de l'image originale avec les points d'intérêts de chacune des imagettes peut poser problème au niveau de la complexité et amener à des cas de figure d'explosion combinatoire. Pour ce faire, nous avons décidé de limiter le nombre de points d'intérêts des imagettes à une 10aine maximum, étant donné le nombre conséquent d'imagettes de la base de données (10000). De plus, ces imagettes vont ensuite être réduites pour rentrer dans le bloc sur lequel elles sont affectées. Elles n'apparaîtront donc qu'à une résolution très petite sur le résultat final.



Image des points d'intérêts obtenus par SIFT et partitionnement de la grille (100x100)

Si l'explosion combinatoire reste gérable et le calcul des descripteurs SIFT (puis la mise en corrélation) peut se réduire, il reste certaines imprécisions notables dans la détermination des résultats d'une image mosaïque. En effet, les techniques de descriptions géométriques permettant d'établir des relations entre des images proches dans une base de données s'appliquent généralement en prenant en compte la totalité de l'image initiale. Ici, on se retrouve à prendre en compte des blocs n'ayant que trop peu d'informations de géométrie. Cependant, plus on agrandit les blocs, et plus on va se retrouver avec des zones couvertes par une seule imagette, et obtenir des résultats pas terribles visuellement.

Ainsi, les techniques de détection et de mise en correspondance de primitives géométriques, telles que des points d'intérêts, ne semblent pas s'appliquer correctement à de tels types de problématiques, du moins pas directement par prise en considération de la meilleure imagette. Elles sont cependant plus utilisées lorsqu'il s'agit de faire du recalage (mise en correspondance de plusieurs images pour aboutir à une image combinant au sens large les informations respectives de chaque image) de photographie pour opérer une cartographie d'une zone avec plusieurs points de vues aériens ou satellites par exemple, ou dans un usage plus courant, pour la création d'images panoramiques avec son téléphone. Plus généralement, ces techniques sont aussi employées dans l'IA pour la détection d'objets.

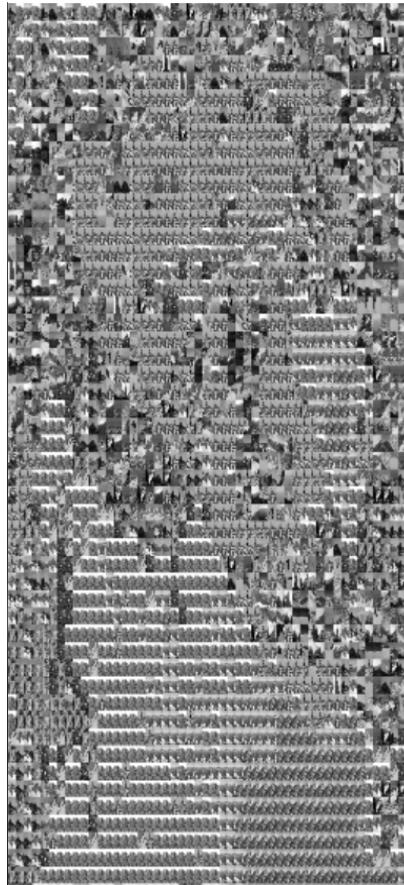


Image mélangeant SIFT et moyenne (100x100)

Conclusion

On peut, avec un partitionnement simple, avoir une abstraction permettant de mettre en œuvre différentes techniques de mosaïque. Il suffit donc de comparer la moyenne, de calculer la distance ou de modifier les couleurs des imagettes pour améliorer le rendu final.

On peut également avoir un partitionnement plus complexe nous permettant d'avoir différentes tailles d'imagettes, comme avec un quadtree.

Pour la mise en application de détections de points d'intérêts puis de mise en corrélation de ces points d'intérêts ensuite, cela permet une analyse géométrique locale de notre image (par bloc), mais ne produit pas de résultats très encourageants.

En effet, il faudrait ensuite mettre en place une phase de recalage des imagettes, pour ne pas seulement les positionner telle qu'elle sur le bloc, mais leur opérer une transformation géométrique adéquate afin d'en plus permettre de les orienter correctement, et ainsi voir se dessiner plus fidèlement les contours de l'image d'origine. Cependant, on sort du cadre classique d'image mosaïque qui a pour but à la fois de reconnaître l'image originale dans le résultat final, mais également la présence de toutes les imagettes qui le compose.

