



Rapport final: Fusion d'exposition

A.U. **Damien Pelissier, Nicodème Gorge**
(Encadrant : Saïd Ladjal)

Table des matières

1	Introduction	1
1.1	Présentation du problème	1
1.2	Présentation de l'article	2
1.3	Objectifs	2
2	Implementation	4
2.1	Méthode Naïve	4
2.1.1	Weight-maps	4
2.1.2	Résultats	5
2.2	Multi-Résolution	5
2.2.1	Pyramide de Laplace	5
2.2.2	Méthode	6
2.2.3	Premiers résultats et incompréhensions	7
2.2.4	Modifications apportées	8
3	Conclusion	10
3.1	Résultats finaux	10
3.2	Pistes d'amélioration	10
3.3	Apprentissages	10

Chapitre 1

Introduction

Ce projet a été développé dans le cadre du cours CSC_4IM01_TP de la filière IMA de Télécom Paris, et s'inspire de l'article suivant : [1]. Vous trouverez tous le code écrit lors de ce projet à l'adresse github suivante Fusion d'exposition.

1.1 Présentation du problème

Cet article a pour objectif de résoudre les problèmes liés à la plage d'exposition des appareils photos, c'est-à-dire réussir à rendre de manière contrastée et détaillée les zones sombres comme les zones lumineuses. En effet, l'œil humain possède une plage dynamique s'étendant sur 20IL (où IL désigne l'indice du lamination) [2], alors que les meilleurs appareils photos ne possèdent qu'une plage dynamique de 12IL. Cela implique que les appareils photos modernes sont incapables de capturer correctement des scènes avec de trop fortes différences de luminosité. Par exemple on peut observer des problèmes de sous-exposition et de sur-exposition 1.1.



(a) Une image sous-exposée (b) Une image sur-exposée

Figure 1.1 – Problèmes liés à la plage dynamique

Il existe des solutions permettant de résoudre au mieux ces problèmes : le photographe peut physiquement rajouter un filtre sur son appareil, comme par exemple un filtre ND dégradé [3], qui va assombrir de façon progressive la scène (ici on modifie la plage dynamique de la scène). Cependant

cela est coûteux et demande un savoir faire propre aux photographes. Une autre solution, cette fois un traitement postérieur à la prise de vue, est la HDR où "High Dynamic Range". Celle-ci consiste, à partir d'une série de plusieurs photos d'une même scène avec différentes expositions, en la création d'une nouvelle image possédant une plage dynamique plus large que celle de l'appareil photo ou des écrans des téléphones par exemple. Cela permet de conserver toutes les informations dans la photo, puis d'afficher l'image en effectuant un "mappage des tons", c'est à dire en compressant les informations lumineuses tout en gardant une apparence naturelle. Voici un exemple d'image calculée par HDR 1.2 Cependant la technologie HDR nécessite de connaître certains paramètres propres au système d'acquisition d'image (par exemple la courbe de réponse de l'appareil), et est également gourmande en calcul, rendant difficile des rendus en temps réel.



Figure 1.2 – Exemple d'image HDR

Dans ce travail, nous nous intéresserons à une méthode alternative qui permet d'éviter ces problèmes, appelée la Fusion d'exposition.

1.2 Présentation de l'article

La méthode par Fusion d'exposition consiste en premier lieu, comme pour la technologie HDR, à prendre une série de photos de la scène avec différents temps d'expositions, puis de fusionner ces images selon certains critères (Cf 1.3).

La technique proposée repose sur des mesures qualitatives simples, comme la saturation, le contraste et une bonne exposition, pour sélectionner et combiner les meilleurs pixels issus de chaque image de la séquence. La saturation est calculée comme la variance entre les trois canaux de couleur RGB, le contraste comme le laplacien de l'image originale et la bonne exposition est obtenue en appliquant un filtre gaussien centré en 0.5 à l'image originale.

On utilise ici une approche multi-résolution basée sur la décomposition pyramidale, qui cherche à assurer une bonne gestion des transitions lumineuses dans les images résultantes, en utilisant plusieurs niveaux de détails.

Cette approche présente des avantages significatifs en termes de simplicité et d'efficacité computationnelle, de plus elle peut également être appliquée sur des images prises avec flash.

1.3 Objectifs

L'objectif de ce travail est d'implémenter l'algorithme décrit dans l'article [1] et de comparer nos résultats avec ceux obtenus par les auteurs de l'article. Nous utilisons pour cela des appareils photos de haute qualité qui nous ont été prêtés par l'association audiovisuelle de l'école Télécom Paris que nous remercions : Comète [4]. Nous implémentons également des fonctions de recadrage, qui sont nécessaires à la fusion correcte des différentes images (l'opérateur qui prend la photo, ou les sujets de la photo, présentent presque toujours un petit mouvement, même si la durée sur laquelle sont prises les photos est très courte). Enfin, nous cherchons ici à avoir un oeil critique sur les résultats obtenus, et les améliorations et solutions à apporter.



(a) Exposure bracketed sequence



(b) Fused result

Figure 1.3 – Exemple de fusion d'exposition issu de l'article

Chapitre 2

Implémentation

Dans cette partie, nous présentons notre travail en suivant son évolution chronologique. Nous nous efforcerons de présenter nos difficultés, résultats et incompréhensions rencontrées à l'époque. On confondra pixel et coefficient d'une matrice carrée.

2.1 Méthode Naïve

2.1.1 Weight-maps

Après plusieurs lectures et l'analyse de l'article, notre première étape a été d'implémenter la méthode naïve telle que présentée dans le papier de recherche. Ce dernier est assez précis sur l'implémentation et nous n'avons pas eu de problème en recréant le filtre de contraste, de saturation et de bonne exposition.

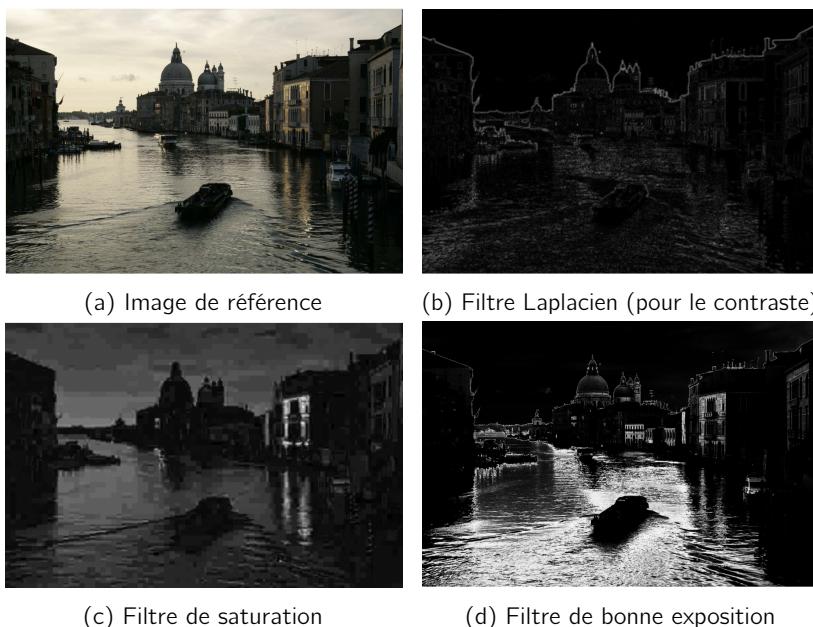


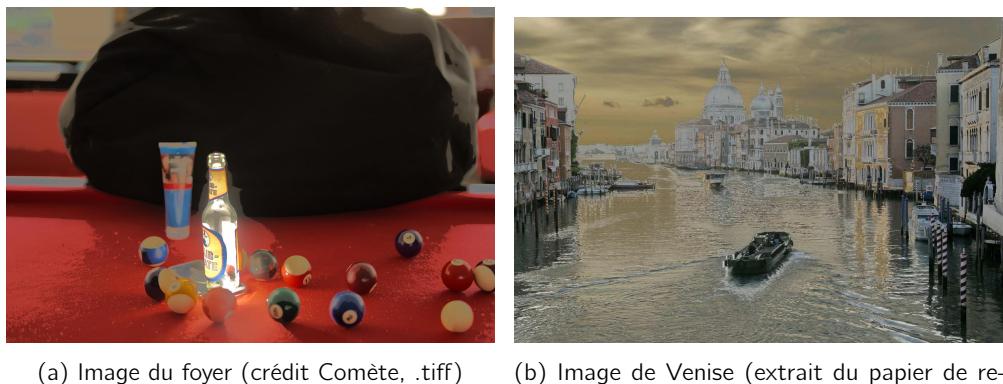
Figure 2.1 – Résultats de nos filtres, ils resteront identiques au long du projet

Remarque : On peut voir sur la weight-map de saturation des défauts comme des carrés grossièrement dessinés. Ces défauts sont dus à l'encodage JPEG. Nous résoudrons ce problème en utilisant nos propres photos à l'avenir.

Le reste de la construction des *weight-maps* (cartes des poids) a été plus compliqué. Lorsque l'on normalise les weight-maps pour que leur somme soit égale à la matrice composée uniquement de 1, on doit diviser par la somme des coefficients. Or, si pour un pixel un filtre donne un résultat nul pour chaque image, alors la somme des coefficients est nulle (inférieure à un $\epsilon \ll 1$). On divise alors par 0, ce qui pose évidemment des problèmes. Dans un premier temps, nous résolvons ce problème en ajoutant un $\delta \approx 10^{-10}$ à chaque weight-map et en l'enlevant après la division. On verra en 2.2.4 une meilleure solution.

2.1.2 Résultats

Comme expliqué dans le papier de recherche, les résultats avec la méthode naïve ne sont pas concluants. On peut voir sur la figure 2.2 que l'on gagne en saturation, mais qu'il y a beaucoup d'artefacts : certains pixels n'ont pas la bonne couleur, et il n'y a pas de transition entre les pixels, ce qui dégrade le rendu de l'image. On peut voir sur la figure 2.2a que le patio visible en haut à gauche est devenu une bouillie de pixels.



(a) Image du foyer (crédit Comète, .tiff) (b) Image de Venise (extrait du papier de recherche, .jpeg)

Figure 2.2 – Résultats de l'implémentation naïve

Cette implémentation ne nous satisfait donc pas et nous allons utiliser une technique de multi-résolution.

Performances

Notre première implémentation avait été faite en utilisant des boucles `for` et d'autres fonctions qui décomposaient nos tableaux `numpy`. Cela nuisait aux performances et on atteignait des temps de calculs de l'ordre de la minute pour la fusion d'exposition avec l'exemple de la figure 2.1. En passant tous les calculs avec les fonctions `numpy`, la même fusion d'exposition s'effectue en moyenne en 0.34s. Ce qui est largement suffisant pour notre utilisation.

2.2 Multi-Résolution

L'approche par multi-résolution n'est que très peu décrite dans l'article, c'est donc le point qui a nécessité le plus de recherches de notre part. L'idée de la multi-résolution réside dans la décomposition de l'image en plusieurs niveaux de détails, en utilisant une pyramide de Laplace.

2.2.1 Pyramide de Laplace

Pour construire cette pyramide, on commence par générer une pyramide gaussienne, en sous-échantillonnant l'image de manière itérative par un facteur de deux après l'application d'un filtre

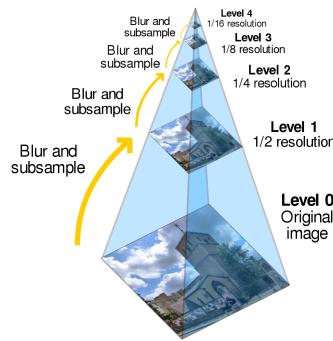


Figure 2.3 – Illustration de la construction d'une pyramide laplacienne © 2023 TheAILearner

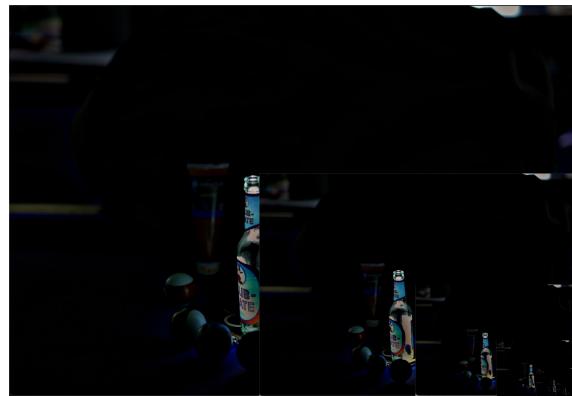


Figure 2.4 – Exemple de pyramide Laplacienne issue de notre implémentation

gaussien. Ensuite, chaque niveau de la pyramide de Laplace est obtenu en calculant la différence entre une image à un niveau donné de la pyramide gaussienne et une version interpolée (agrandie) de l'image au niveau immédiatement inférieur. Cette décomposition en niveaux permet d'isoler les détails spécifiques à différentes échelles de résolution 2.3. On peut voir ce que l'algorithme retourne 2.4

À partir de cette étape, on ne travaille plus simplement sur des images, mais sur des pyramides d'images, ce qui rend rapidement plus difficile la compréhension des structures utilisées. Afin de pouvoir s'y retrouver, nous avons réalisé un schéma liant les différents objets entre eux 2.5.

Nous devons désormais appliquer le même processus que précédemment pour chaque étage de la pyramide avant de reconstruire celle-ci et de fusionner les images. Nous expliquons dans cette section comment nous avons réalisé cette nouvelle implémentation.

2.2.2 Méthode

Pour réutiliser au maximum le code que nous avons écrit précédemment, nous allons simplement utiliser les mêmes fonctions qu'avant via des *wrappers*. Cela demande une réorganisation des images non par pyramide mais en vecteurs contenant les mêmes étages dans différentes pyramides.

```

1 def get_wms(imgs, power_coef, show=False):
2     [...]
3     wms = []
4     for img in imgs:
5         wms.append(get_wm(img, power_coef, show=show, [...]))
6     return wms

```

Listing 2.1 – Exemple d'un wrapper

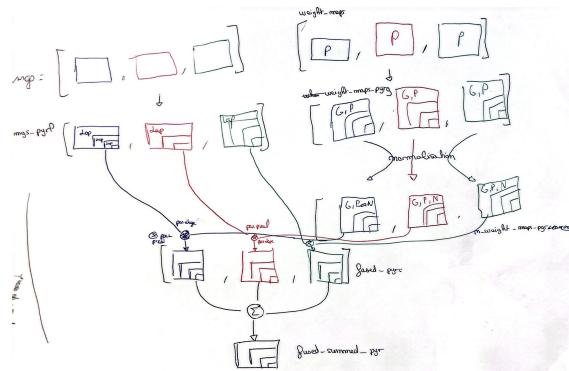


Figure 2.5 – Schéma du problème

2.2.3 Premiers résultats et incompréhensions

La figure 2.6 illustre plusieurs résultats issus de notre implémentation à l'époque. Chaque photo illustre un problème rencontré. Il nous a fallu à chaque fois tester beaucoup de choses pour comprendre ce qui était des bugs d'implémentation, ou des erreurs de logiques, ou encore des comportements des pyramides laplaciennes indésirables.

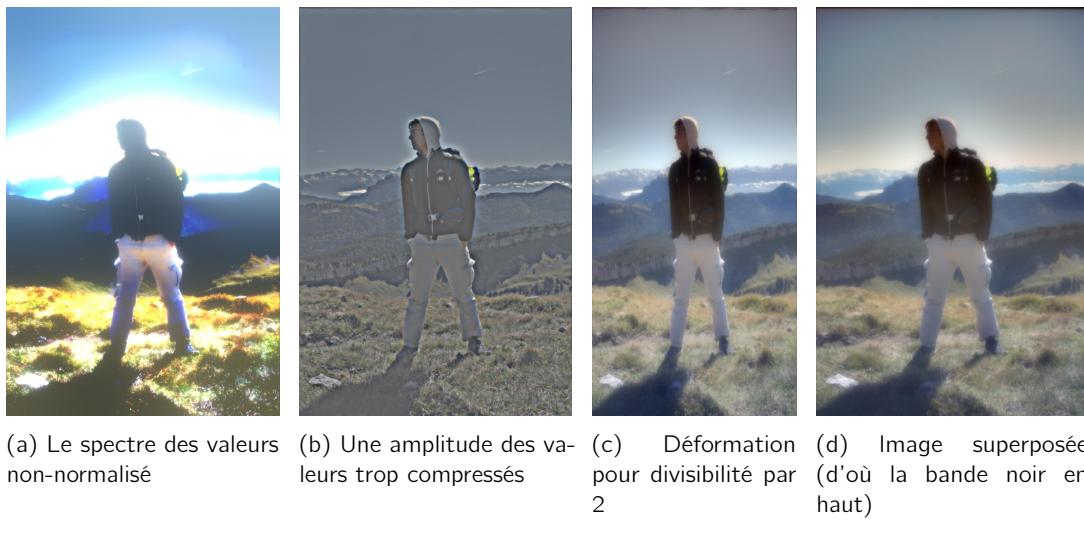


Figure 2.6 – 4 Exemples de problèmes rencontrés lors de l'implémentation de la technique multi-résolution

Pour répondre à tous ces problèmes qui étaient difficilement traçables, nous avons mis en place un framework de tests avec `pytest` couplé à des décorateurs vérifiant l'entrée et la sortie de chacune de nos fonctions (leur type, si c'est normalisé). Ainsi je vais citer les solutions pour chaque problème illustré dans la figure 2.6.

- (a) Même si tous les étages de la pyramide sont normalisés, après l'application des filtres et des weight-maps normalisées, l'image issue la fusion de la pyramide laplacienne n'est pas normalisée. Dans l'exemple de la figure les valeurs appartiennent à $[-4; 6]$. Cela explique pourquoi l'image est saturée. Il faut donc réduire la dynamique de l'image entre 0 et 1
- (b) Une mauvaise implémentation de la solution précédente conduisait à une dynamique trop réduite, et donc à une image désaturée.

- (c) Pour utiliser une pyramide laplacienne à n étages, il faut que la largeur et la hauteur de l'image soit divisible par 2^n (valuation 2-adique minimale désirée). Pour s'assurer que notre photo soit bien divisible par 2^n , on redimensionne la photo. Si on décide de prendre n trop élevé, la déformation est alors visible à l'œil nu. On est donc limité dans la taille de notre pyramide laplacienne.
- (d) Comme la prise a été faite en condition de fort vent, et que le sujet est vivant et bouge inévitablement, l'image résultante de la fusion est donc flou. On peut voir ce flou sur les trois premières images. On implémente donc une transformation SIFT comme vu en cours pour minimiser le déplacement entre les photos. On voit son utilisation à la barre noire en haut de l'image qui révèle le décalage effectué par l'algorithme. Le résultat est donc moins flou mais on utilisera tout de même un algorithme de `unsharp` (on floute l'image, et on soustrait à l'image originale la version floutée) pour déflouter le résultat. L'image utilisée en 3.1 a été prise à la fin du projet sur un trépier pour empêcher tout flou.

2.2.4 Modifications apportées

Après avoir résolu les problèmes, l'image obtenue (qu'on notera \mathcal{A}) est encore loin du résultat présenté dans le papier de recherche. Nous avons donc implémenté 2 nouveautés par rapport au papier.

Spécification d'histogramme et CLAHE

Comme la dynamique de \mathcal{A} a subi un *contrast stretching*, l'image n'a plus les mêmes caractéristiques au niveau de l'histogramme cumulé. L'idée est donc d'utiliser une spécification d'histogramme sur les canaux de saturation et de valeur. On peut voir le résultat sur la figure 2.7

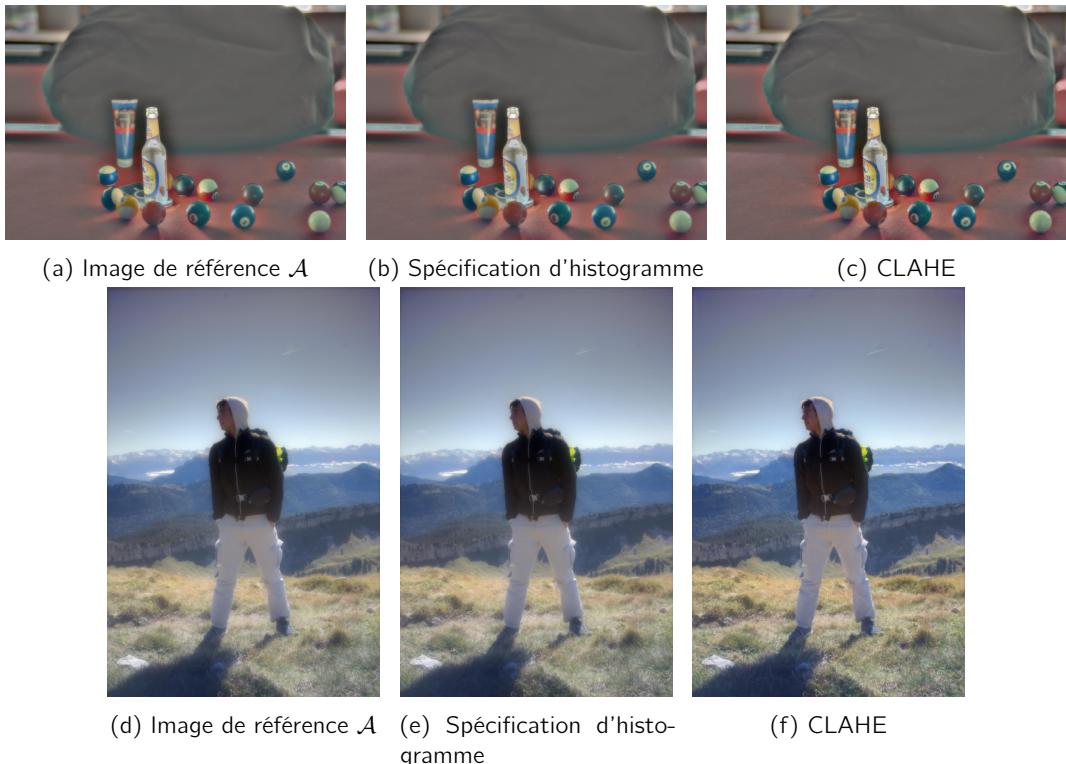


Figure 2.7 – Effet de l'application des recolorations

Remarque : On précise la légende

- La spécification d'histogramme est par rapport à l'histogramme moyen des images de départ

- L'algorithme CLAHE signifie *Contrast Limited Adaptive Histogram Equalization*. Il améliore le contraste des images en limitant l'amplification du bruit localement.

Désactivation des filtres non discriminants

L'implémentation de la normalisation en 2.1.1 présente plusieurs défauts que nous avons essayé de corriger. Choisissons un pixel. Si pour un filtre, la valeur de chacune de ses weight-maps est très faible, cela veut dire que pour ce pixel, le filtre n'est jamais satisfait. Comme on multiplie les weight-maps des filtres entre elles, toutes les weight maps seront quasiment nulles en ce pixel. Ainsi, l'algorithme va fusionner de manière uniforme les valeurs de ce pixel, même si les autres filtres sont discriminants. Pour éviter cette perte d'information, on crée un masque par filtre. Si le filtre donne des valeurs trop faibles (inférieur à un seuil η), on change la valeur du masque. Lors de la fusion des weight-maps des filtres on oubliera celles dont le masque est changé pour le pixel.

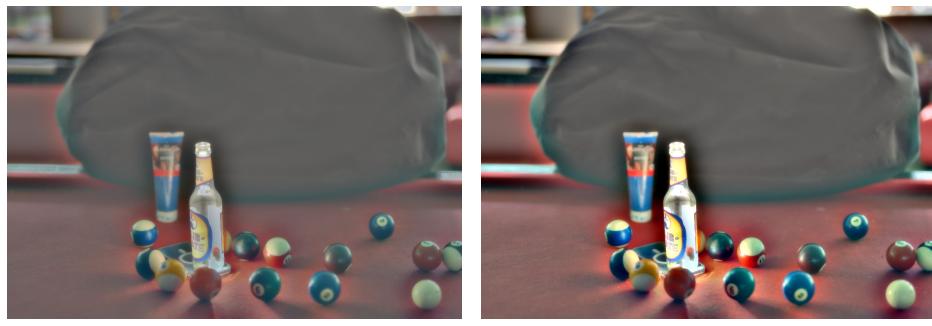


Figure 2.8 – Comparaison des images avant et après désactivation des filtres non discriminants

Chapitre 3

Conclusion

3.1 Résultats finaux

Dans la figure 3.1 on présente les résultats finaux obtenus en faisant varier les coefficients en puissance pour le contraste, la saturation et l'indice d'exposition. Bien qu'il existe quelques différences entre les différentes images (avec une assez bonne qualité on voit la différence entre 3.1a et 3.1b au niveau de la boule blanche en bas à droite), ces dernières sont minimales. Ainsi deux conclusions sont envisageables, soit l'influence des ces paramètres est très faible, soit une erreur réside quelque part dans notre implémentation.

3.2 Pistes d'amélioration

Hormis les points mentionnés dans les parties précédentes, il serait intéressant de chercher à supprimer le phénomène de "bloom" que l'on peut observer sur l'image (i.e. la diffusion de la lumière autour des zones très lumineuses d'une scène), par exemple on peut l'observer sur cette image 3.2 (autour de la bouteille). Ce phénomène est lié à l'implémentation de la fusion d'exposition, et potentiellement empiré par le post-traitement permettant d'améliorer la netteté (sharpening).

Nous n'avons pas eu l'occasion de tester notre algorithme pour la fusion de photos prises avec flash et sans flash, qui sont sensées être gérées correctement par l'algorithme.

3.3 Apprentissages

Un des aspects qui nous a le plus marqué lors de ce projet est l'intérêt d'utiliser une approche de développement "test driven", c'est à dire de créer des tests pour chaque fonction (même avant de créer les fonctions elles-même!). Nous avons initialement codé toute la méthode naïve en faisant très peu de tests, cela a résulté en de nombreux problèmes au niveau des objets manipulés et de la compréhension globale du fonctionnement des fonctions. Par exemple, lorsque nous sommes passés de la fusion de simples images, à la fusion d'images au sein d'une pyramide, cela a donné lieu à des structures plus complexes. Nous avons alors rencontré de nombreux problèmes, ce qui nous a amené à coder deux fonctions : une première `inspect_list_structure` permettant d'avoir des informations sur la structure de l'objet manipulé 3.1.

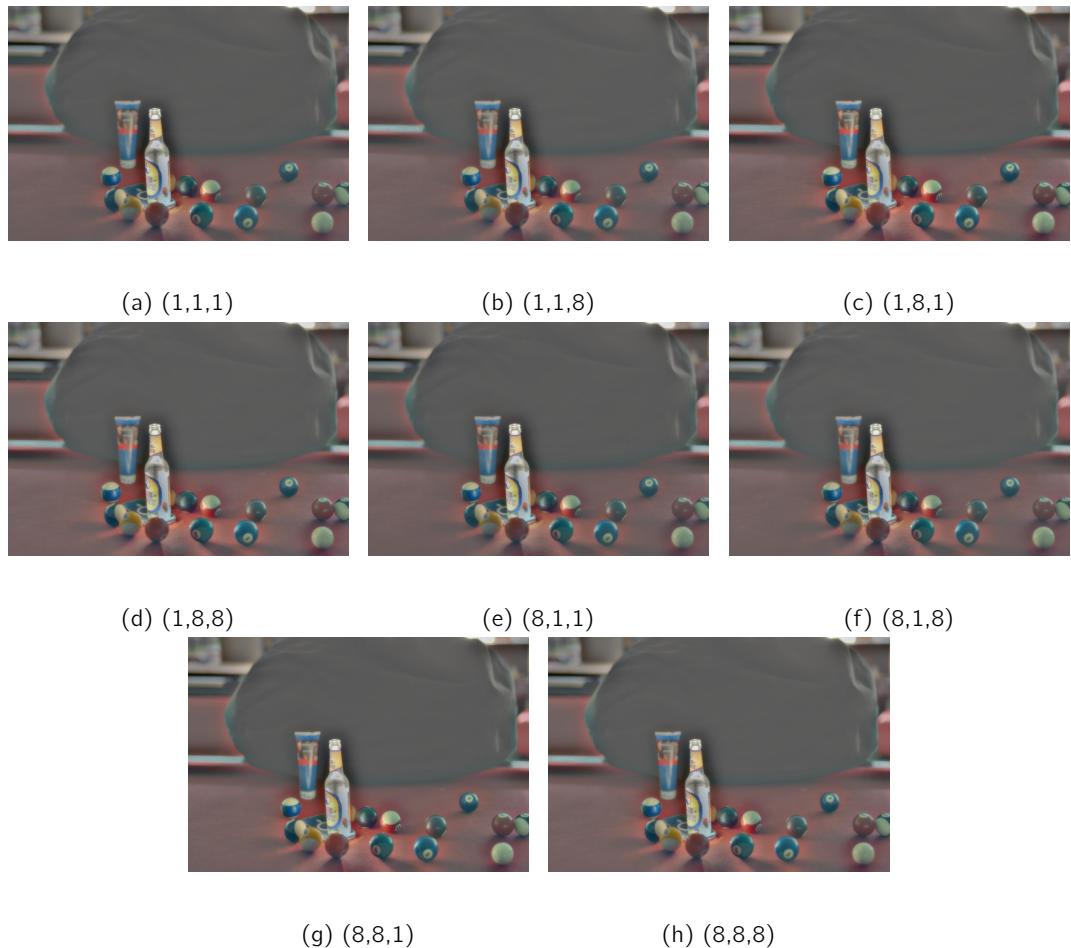


Figure 3.1 – Comparaison des résultats en fonction des coefficients (contraste, saturation, exposition)



Figure 3.2 – Exemple de bloom, autour de la bouteille

```

1 example_list = [
2     np.random.rand(64, 64, 3).astype(np.float64),
3     [np.random.rand(28, 28).astype(np.float64), np.random.rand(28, 28).astype(np
4         .float64)],
5     42]
6 inspect_list_structure(example_list)

```

Listing 3.1 – Exemple d'utilisation de `inspect_list_structure`

Listing 3.2 – Résultat de l'exécution

```

List of length 3
Element 0
Numpy Array with shape (64, 64, 3), dtype float64
Element 1
List of length 2
Element 0
Numpy Array with shape (28, 28), dtype float64
Element 1
Numpy Array with shape (28, 28), dtype float64
Element 2
int: 42

```

Et une deuxième fonction, un décorateur, permettant de vérifier à chaque utilisation d'une fonction que le format de l'entrée est le bon. Nous avons également implémenté d'autres décorateurs pour notamment vérifier la normalisation des images utilisées. Enfin, comme mentionné précédemment nous avons utilisé le framework `pytest` pour tester notre code. Ainsi tester systématiquement chaque fonction nous a permis de ne pas nous retrouver avec des problèmes trop complexes et difficiles à débuguer, pour se concentrer d'avantage sur les problèmes liés à l'implémentation en elle-même, ainsi que de bien visualiser chaque objet.

Bibliographie

- [1] Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion. *Pacific Graphics*, 2007.
- [2] Ifolor. Plage dynamique en photographie numérique. <https://www.ifolor.ch/fr/inspirations/plage-dynamique>.
- [3] Vivre de-la photo. Qu'est-ce que la plage dynamique en photo et comment l'améliorer ? <https://vivre-de-la-photo.fr/quest-ce-que-la-plage-dynamique-en-photo-et-comment-lameliorer/>.
- [4] Jean Groeninger. L'association de photos et vidéos de télécom paris. <https://www.comete-tp.fr/>.