

MARCHANDOT
SAHEB

Benoît
Zakary

24 NOVEMBRE 2020

WATERPIXELS

RAPPORT PROJET IMA201



Sommaire

Introduction	3
1 Les étapes de l'implémentation de waterpixels	4
2 Implémentation des fonctions	6
2.1 Les gradients	6
2.2 La création de cellules	7
2.3 Marquage des minima du gradient pour chaque cellule	10
2.4 Carte de distance	13
2.5 Normalisation du gradient	14
2.6 Ligne de partage des eaux	15
3 Interprétation des résultats	16
Conclusion	22
Bibliographie	23

Introduction

Les waterpixels sont une méthode de segmentation d'une image se basant sur le principe de ligne de partages des eaux (Watershed). De cette méthode résulte la création de Super Pixels, permettant notamment de travailler sur différentes régions homogènes de l'espace. Cette méthode se base sur le lien entre l'étude de la régularisation spatiale du gradient, et le bon choix de marqueurs pour appliquer l'algorithme du watershed.

Ce projet s'appuie sur des notions que nous avons travaillées en cours d'IMA201 et en TP, mais a aussi fait appel à des compétences que nous n'avions pas encore acquises. Il nous a donc permis de renforcer nos acquis, et de découvrir de nouveaux outils et méthodes de calcul (comme les cartes de distances par exemple).

Au cours de ce projet, nous avons été accompagné par Florence Tupin, enseignant-chercheur à Télécom Paris, que nous rencontrions toutes les deux semaines.



FIGURE 1 – Waterpixels présentés dans l'article de recherche étudié

1 Les étapes de l'implémentation de waterpixels

Calcul du gradient de l'image

Cette étape consiste en le calcul simple du gradient de l'image. L'article explique que le type d'image peut influencer notre choix du calcul de ce dernier. Nous avons fait le choix d'utiliser deux opérateurs vus en travaux pratique, le gradient de Sobel et le gradient morphologique. Ce gradient sera plus tard utiliser pour choisir les souches de l'algorithme de ligne de partage des eaux, et pour obtenir un gradient normalisé.

Définition de cellules

La définition des différentes cellules se base sur la distance et la taille de la bande séparante entre chacune d'elles. Nous avons choisi de suivre l'exemple de l'article et d'opter pour une définition de cellules hexagonales, dont la création explicitée ultérieurement, se base sur une itération de dilatations.

Sélection des marqueurs

Pour le choix des marqueurs, nous devons assigner dans chaque cellule une unique composante connexe, que nous définissons de la manière suivante (de manière itérative sur chaque cellule) : -Calcul de la valeur minimale du gradient au sein de la cellule. S'il n'y en a pas et que le gradient est homogène, on choisit comme marqueur le centre de la cellule.

-Sinon, on marque de tous les points de la cellule portant cette valeur de gradient, ce sont tous les minima du gradient. Finalement, on choisit la composante connexe des minima du gradient la plus grande.

Ces marqueurs que l'on notera dans la suite M_i , seront utilisés par la suite pour l'algorithme du watershed.

Régularisation spatiale du gradient

Pour réaliser la régularisation spatiale du gradient, il faut d'abord passer par une carte de distance créée à partir des marqueurs définis précédemment. Cette carte de distance donne en chaque point de l'image la distance à l'objet (ici le marqueur) le plus proche. On peut également créer cette distance à partir des centres des cellules, modifiant dans ce cas le waterpixel final. Ensuite, la régularisation se fait selon la formule donnée par l'article, $g_{reg} = g + k.d$, où g est

le gradient, d la distance rapportée à la taille d'une cellule, et k est un facteur de normalisation (si $k \rightarrow \infty$, on a les cellules de Voronoï, si $k = 0$, c'est le watershed original).

Application de l'algorithme du Watershed

Finalement, il ne reste plus que d'appliquer l'algorithme du watershed à l'image, avec le gradient régularisé (selon la carte de distance), et en prenant comme marqueurs les minima des gradients M_i de chaque cellule.

Voici un exemple illustré donné dans l'article :

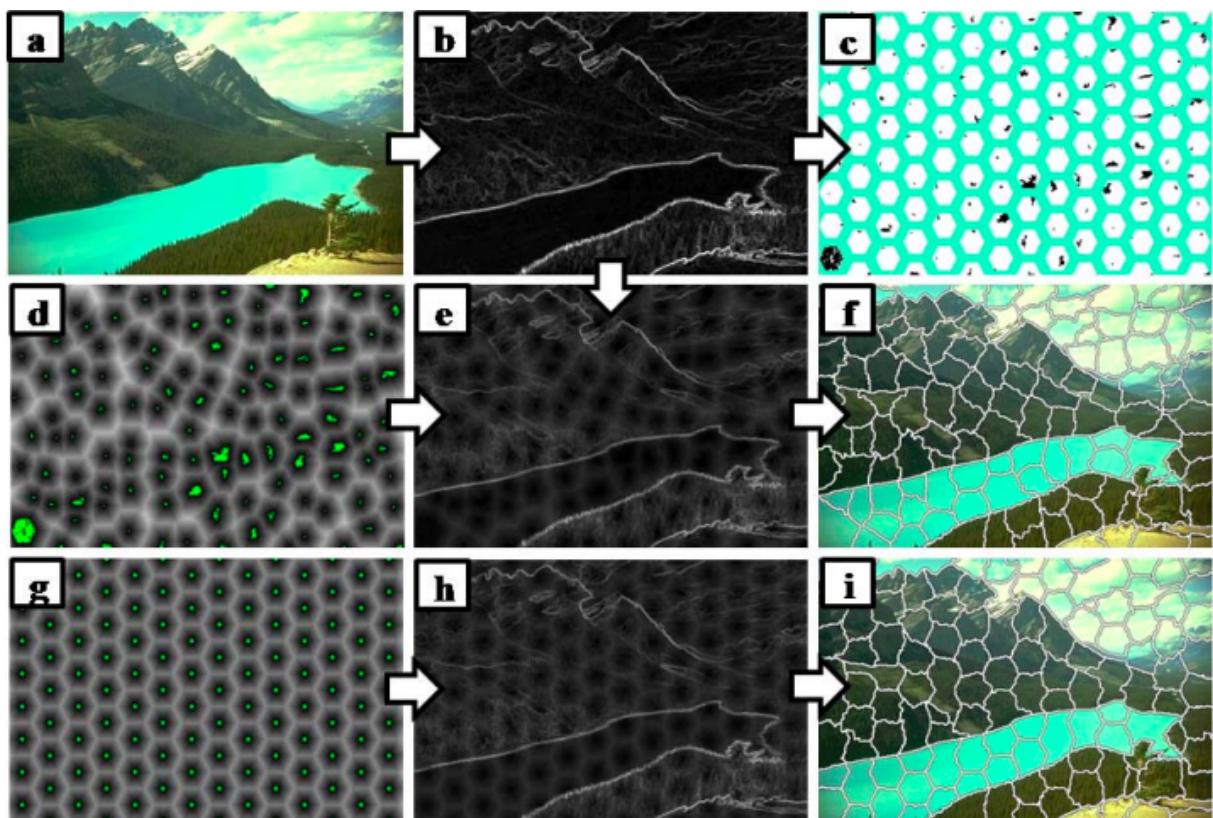


FIGURE 2 – Création de cellules pour l'image bat.bmp

2 Implémentation des fonctions

2.1 Les gradients

Plusieurs gradients ont été utilisés. Le gradient de Deriche, le gradient de Sobel, ainsi que le gradient morphologique. Dès le début, nous avons choisi de ne conserver que le gradient de Sobel, ainsi que le gradient morphologique car ils correspondent le mieux à nos critères. Ces derniers sont établis sur la sensibilité du gradient, la robustesse au bruit, et surtout la bonne restitution des contours. Le gradient de Sobel prend en paramètre deux réels qui permettent d'ajuster la sensibilité au bruit ($\text{seuilnorme}, \sigma$). Quant au gradient morphologique, il correspond assez bien à nos attentes, car il détecte très bien les contours, et est assez robuste au bruit. C'est d'ailleurs celui-ci qui apporte les meilleurs résultats sans avoir besoin de jouer sur des paramètres supplémentaires.

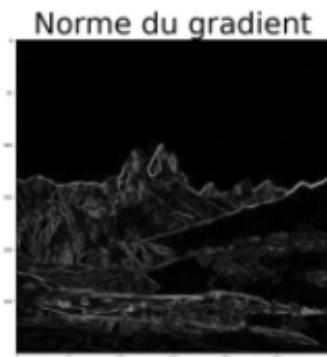


FIGURE 3 – Gradient morphologique pour montagne.png

2.2 La création de cellules

Contre toute attente, c'est cette partie du projet qui nous a donné le plus de fil à retordre. Dans un premier temps, nous avons envisagé de créer des cellules carrées, car nous n'avions pas d'autre idées concernant une création de cellules en partant de rien. Nous avons donc crée un quadrillage, mais celui-ci s'adaptait très mal aux différentes tailles d'image, et nous n'obtenions rien de satisfaisant.

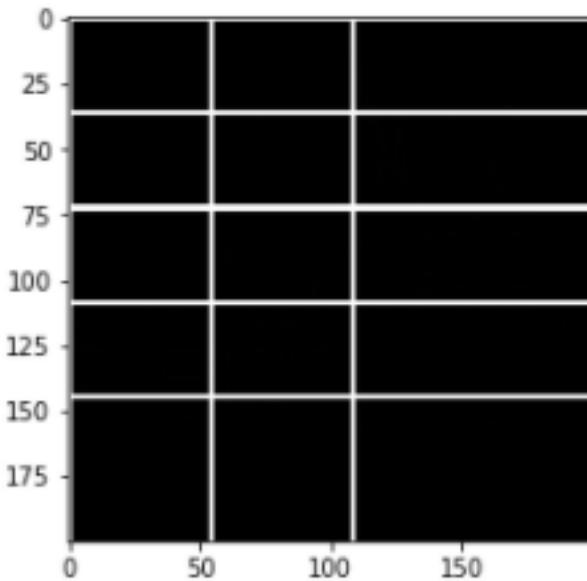


FIGURE 4 – Premier essai de création de cellules avec des quadrillages

Puis il nous a été suggéré d'itérer des dilatations, en ayant préalablement fait un pavage hexagonal (insérer des points blancs dans une matrice nulle selon le pavage voulu). Pour cette dilatation, nous avons choisi l'élément structurant suivant :

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

La fonction `creation_cellule` prend donc en argument l'image dont on veut créer un masque de cellules, `taille_cellule` la taille de cellule que l'on souhaite obtenir en sortie, ainsi que `bande_separante_min` la taille de l'espace entre les cellules (en pixels). Le résultat obtenu était maintenant particulièrement satisfaisant (du moins pour l'instant).

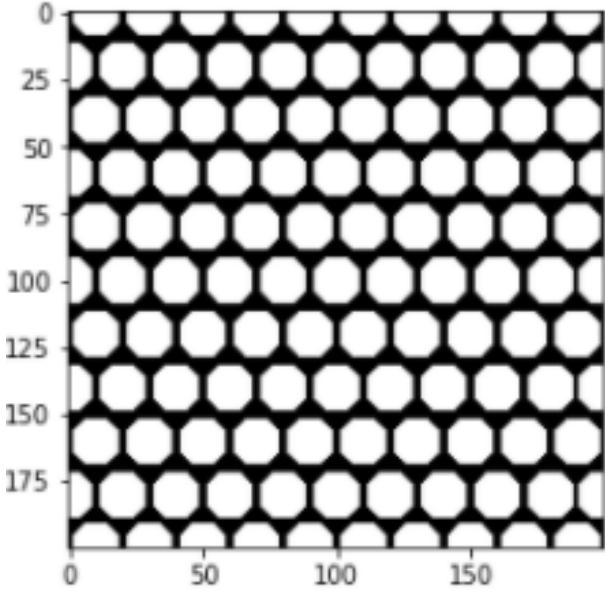


FIGURE 5 – Création de cellules pour l'image bat.bmp

Par la suite, des problèmes sont survenus lorsqu'il est question de traiter des images bien plus grandes. Certes, le traitement d'images plus grande a aussi suscité des problèmes dans le calcul du gradient, mais les résultats observés étaient déstabilisants. Le problème est le suivant :

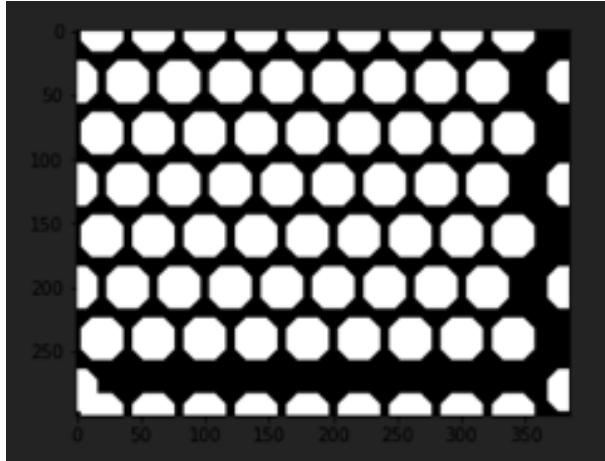


FIGURE 6 – Problème lors de la création de cellules d'une image qui n'est pas carrée

Ceci vient précisément du fait que on parcourt le masque de 0 à taille_cellule . $\left\lfloor \frac{n}{\text{taille_cellule}} \right\rfloor$ où $n = k * \text{taille_cellule} + r$, c'est-à-dire le parcours se fait de 0 à $n - r$, où $r \geq 1$. En conséquence, il manque à droite une colonne, dans laquelle on ne peut pas insérer d'autre cellule sans qu'il n'y ait chevauchement (la dernière colonne était traitée séparément, pour des raisons pratiques). Et ceci se produit de même sur les lignes et sur les colonnes. Une alternative pourrait être de créer ces cellules récursivement : nous avons tenté d'écrire un tel algorithme,

mais de nombreuses complications se sont vite présentées, et nous avons finalement économisé notre temps pour autre chose. Enfin, pour avoir une bonne répartition de ces cellules, il suffit de tester pour plusieurs paramètres taille_cellule différent, afin de tomber sur quelque chose de satisfaisant. Enfin, le choix du paramètre taille_cellule reste discutable. Lorsque l'image est petite (de taille 200 * 200 par exemple) 20 pixels est une taille appropriée. Mais lorsque l'image est très grande (1000x1000), ce n'est plus approprié, et faut choisir une taille de cellule de 100 par exemple. En plus de cela s'ajoute le coût temporel qui devient bien trop important pour un grand nombre de cellules, notamment dans la fonction min_grad. En conséquence il faut tester pour chaque image la fonction creation_cellule pour déterminer quelle valeur de taille_cellule convient le mieux.

2.3 Marquage des minima du gradient pour chaque cellule

Après avoir créé les différentes cellules, l'article nous dit de marquer pour chacune les pixels correspondant aux minima du gradient. Pour ce faire, il a fallu dans un premier temps étiqueter en composante connexe les différentes cellules de manière à pouvoir travailler successivement dans chacune. L'utilisation de la fonction label de skimage permet de faire ceci, renvoyant un tableau M de même dimension que l'image, où $M[i][j]$ correspond à l'indice de la composante connexe (donc de la cellule) à laquelle le pixel de l'image appartient.

Une fois ce premier étiquetage effectué, pour chaque cellule, nous avons créé une image temporaire de la même dimension que l'image d'origine, mais ne comportant que la cellule étudiée, pour laquelle nous affichons la valeur du gradient.

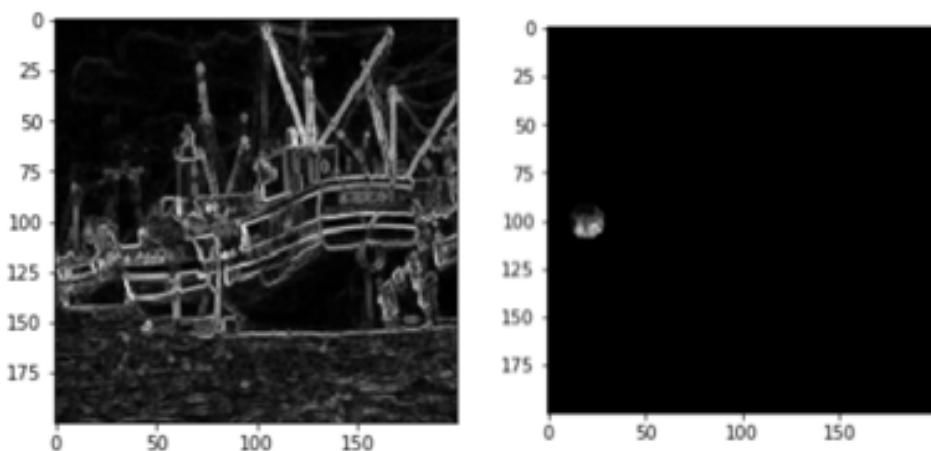


FIGURE 7 – Image du gradient de la cellule concernée uniquement

A partir de cette nouvelle image, nous extrayons le minimum du gradient dans la cellule, nous permettant ainsi de comparer toutes les valeurs du gradient dans la cellule et d'en extraire tous les pixels correspondant à ce minimum du gradient.

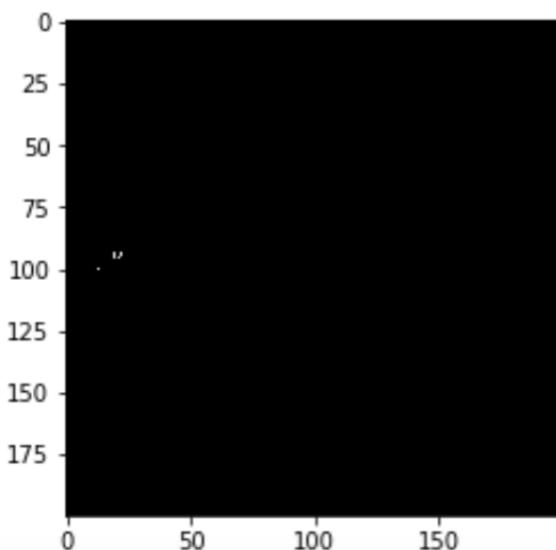


FIGURE 8 – Minimum du gradient de la cellule concernée uniquement

Finalement, pour choisir quels pixels utiliser comme marqueurs, nous avons pris le choix de ne retenir que la composante connexe la plus grande grâce à la fonction `mostcommon` de Counter, donnant les indices des pixels correspondant à la composante connexe la plus grande (avec l'indice apparaissant le plus de fois). Dans le cas où le gradient vaut 0 dans toute la cellule, nous prenons comme marqueur le centre de celle-ci.

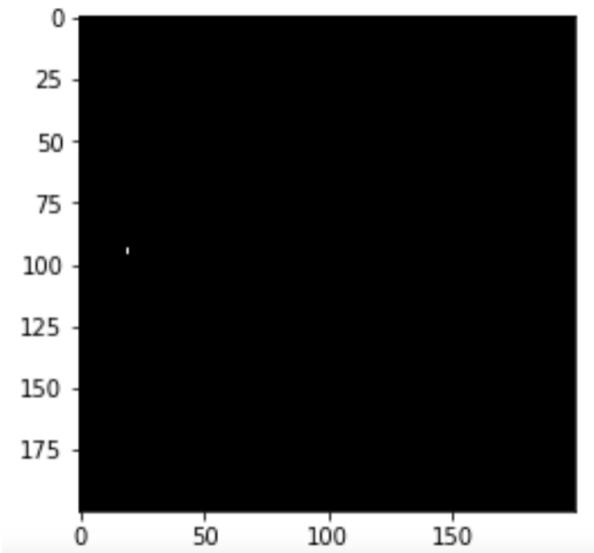


FIGURE 9 – Composante connexe la plus grande pour le minimum du gradient de la cellule précédente

Cette méthode permet donc de donner les composantes connexes correspondant aux minimum du gradient pour chaque cellule.

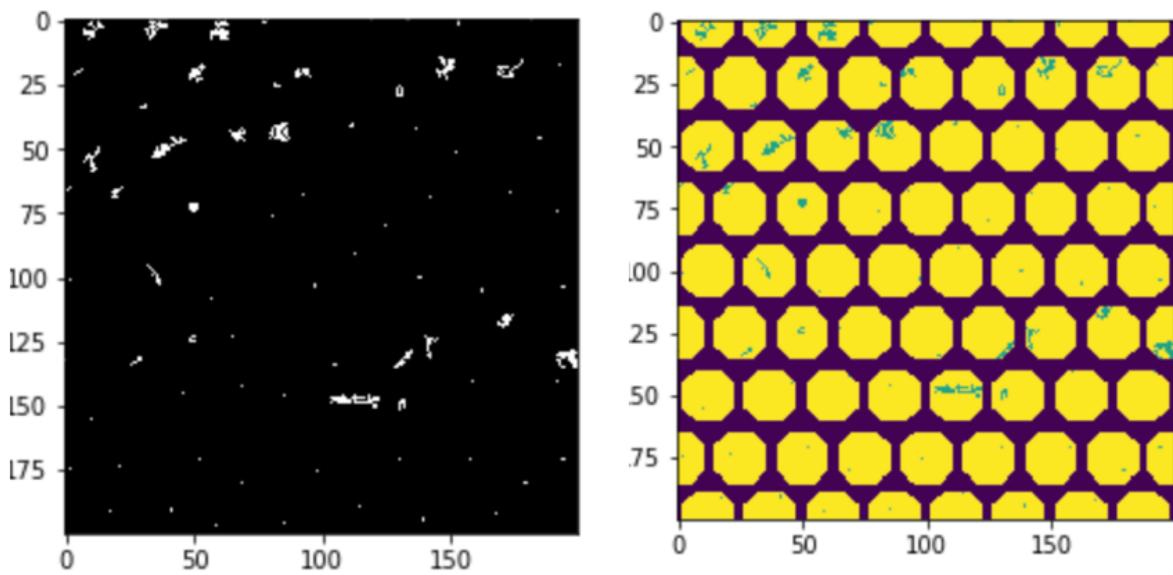


FIGURE 10 – À gauche le gradient de l'image, et à droite, les plus grandes composantes connexes du minimum du gradient, pour chaque cellule

Amélioration de la méthode

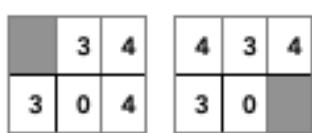
Cette méthode, bien qu'efficace, est extrêmement coûteuse. En effet, elle demande de recréer une image pour chaque cellule, et de la parcourir en entier pour recréer une image avec les pixels correspondant aux minimum, pour ensuite étiqueter une nouvelle fois en composante connexe et de trouver la plus grande. Cela crée plusieurs images de taille $N \times M$ et les parcourt plusieurs fois. Pour avoir une méthode marchant sur le même principe mais moins coûteuse, nous pourrions travailler non pas sur une nouvelle image de même dimensions que l'originale et ne comportant que la cellule étudiée, mais sur une image plus petite correspondant à un découpage de l'image de base autour de la cellule. Cela permettrait de réduire considérablement le nombre d'opérations à effectuer. Il faudrait alors calculer en fonction de la taille de la cellule la manière de découper l'image convenablement. Cette amélioration pourrait permettre d'obtenir une complexité linéaire comme mentionnée dans l'article.

2.4 Carte de distance

Cette partie de code n'a pas posé beaucoup de problème. Les cartes de distance d'une image à deux niveaux de gris se calcule en parcours l'image dans un sens, puis dans l'autre, et pour chaque pixel il faut appliquer un masque. Au départ, l'image est initialisée de la manière suivante : tous les pixels noir (ce qui composent les trous) prennent tous une valeur infinie, et les pixels des objets prennent une valeur nulle. Ainsi, lors du parcours de l'image, dès que l'on rencontre un objet, le masque remplace le pixel par le minimum entre les pixels du masque, et le pixel sur lequel on travaille. Ainsi, plus on s'éloigne de l'objet, plus la valeur des pixels augmente. La fonction `distance_map` prend en argument deux choses : l'image et le masque. Il est possible d'utiliser un des trois masques suivant :



Les deux demi-masques en 4-connexité



Les deux demi-masques en 8-connexité 3-4



Les deux demi-masques en 8-connexité 5-7-11

Les différences sont notables, et dans la suite, nous utiliserons toujours les demis masques 5-7-11 en 8-connexité.

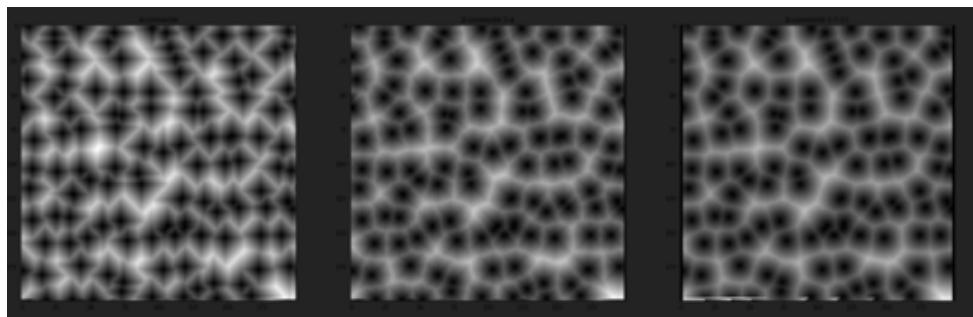


FIGURE 11 – Différentes cartes de distances calculée avec les 3 masques différents 4-connexité, 8-connexité 3-4 et 5-7-11 (de gauche à droite)

L'utilisation de la carte des distances est essentielle pour la fonction de normalisation du gradient.

2.5 Normalisation du gradient

La normalisation du gradient s'écrit comme $\text{gradient} + \frac{2}{\text{taille_cellule}} \cdot k \cdot \text{distance_map}$ où k est un entier à définir. En conséquence, on obtient l'image ci-dessous (pour bat.bmp par exemple). On remarquera que lorsque $k=0$, on obtient le gradient initial en sortie, et lorsque k tend vers l'infini, on obtient la carte de distance en sortie.

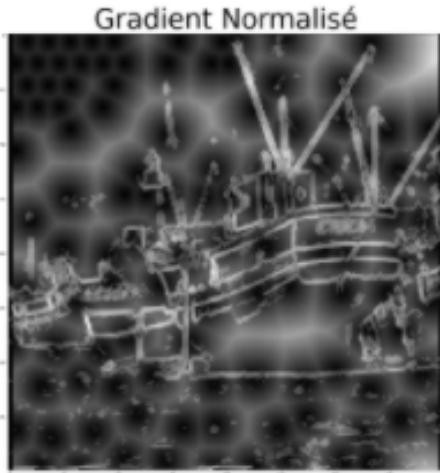


FIGURE 12 – Gradient normalisé pour bat.bmp

On remarque notamment que lorsque $k=0$, on obtient le gradient initial en sortie, et lorsque k tend vers l'infini, on obtient la carte de distance en sortie. Ce gradient normalisé, est une manière de pondérer l'importance des zones du gradient à prendre en compte dans le calcul des lignes de partage des eaux.

Par la suite, lors de l'implémentation des waterpixels, le paramètre k est très important à faire varier, pour pouvoir maîtriser un dessin des pixels trop précis sur les contours, ou au contraire, un dessin des pixels qui ne respecte pas du tout les contours.

En reprenant l'exemple de bat.bmp :

Ces résultats illustrent bien la notion expliquée ci-dessus : lorsque k tend vers une valeur infini, les contours ne sont pas dessinés, et à l'inverse lorsque k tend vers 0, seuls les contours sont pris en compte, ce qui rend difficile le calcul des lignes de partage des eaux.

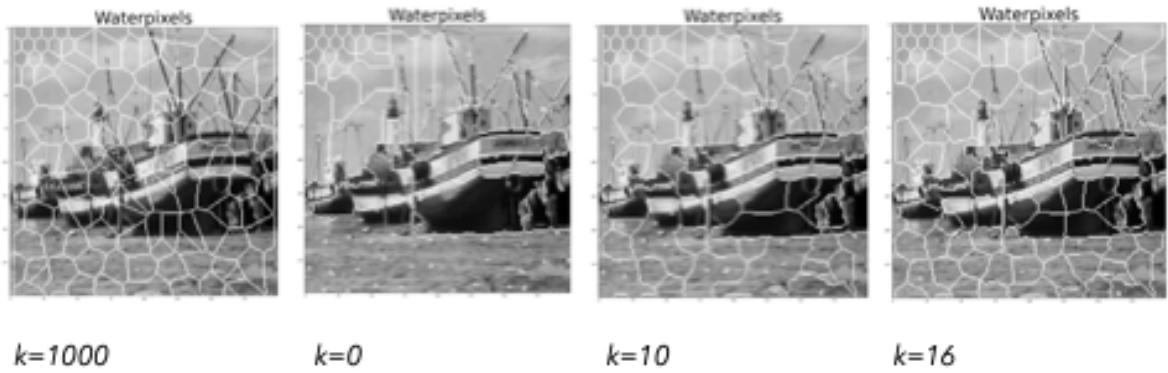


FIGURE 13 – Waterpixels de bat.bmp pour différents k

2.6 Ligne de partage des eaux

Après le calcul des cartes de distance, du gradient normalisé, et la recherche du minimum du gradient, il faut calculer les lignes de partage des eaux. Comme le suggère l'article, ce calcul se fait en utilisant la fonction `morpho.watershed()` sur le gradient normalisé, avec les marqueurs de `min_grad`. Ce calcul dépend simplement du `min_grad`, et du gradient.

3 Interprétation des résultats

Il est important de noter que tous les résultats obtenus dépendent de la qualité de l'image analysée, de sa taille, du gradient utilisé, la taille des cellules, ainsi que le facteur de normalisation du gradient.

Même au sein de notre binôme, nous avons observé des résultats différents pour la même image, puisque nous avons un notebook différent, des mêmes images téléchargées depuis des plateformes différentes etc... Il a été décidé que nous ne ferions des tests qu'avec des images ne dépassant pas une taille de $500 * 500$ car au-delà, les temps de calcul sont bien trop importants ; les complexités de la plupart des fonctions sont en $O(n^2)$.

Tout comme dans l'article [1], nous présenterons nos résultats sous la forme de 9 images représentatives, dans l'ordre :

- Image originale
- Norme du gradient
- Minimum de gradient par cellule
- Carte de distance du minimum du gradient
- Gradient Normalisé
- Waterpixels
- Carte de distance du centre des cellules
- Gradient Normalisé avec le centre des cellules
- Waterpixels en prenant le centre des cellules comme marqueurs

Découpage du *ciel*

De manière générale, les waterpixels, prenant le minimum du gradient comme marqueurs (i.e lorsque les M_i sont les marqueurs) pour le watershed sont très efficaces pour restituer les contours là où le gradient est turbulent. Par contre, là où le gradient est faible, c'est-à-dire pour le *ciel*, le watershed de marqueurs M_i n'est pas très pertinent, puisque des minima de gradient sont tout de même détectés. On observe cela sur les figures suivantes :

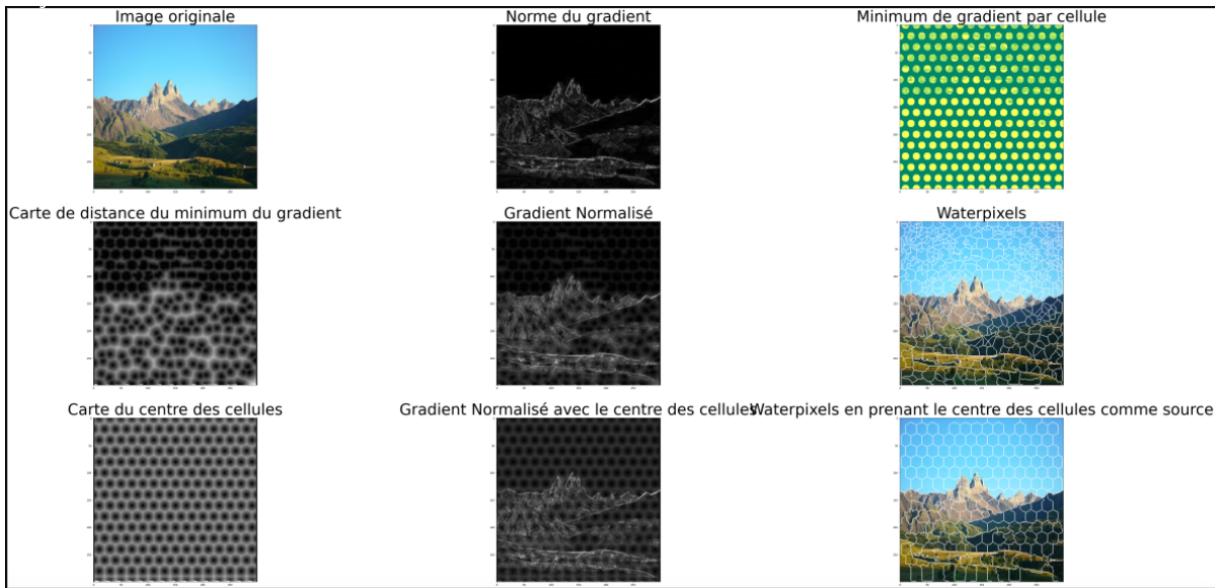


FIGURE 14 – Pour montagne.png, le ciel est assez mal découpé pour le premier watershed, alors que les contours sont très bien restitués là où le gradient est turbulent ($k=10$ ici)

En effet, il serait souhaitable que dans le *ciel*, notre algorithme préfère prendre un marqueur O_i (centre de cellule) que M_i . Cette dernière photo laisse présager que pour une image animée,

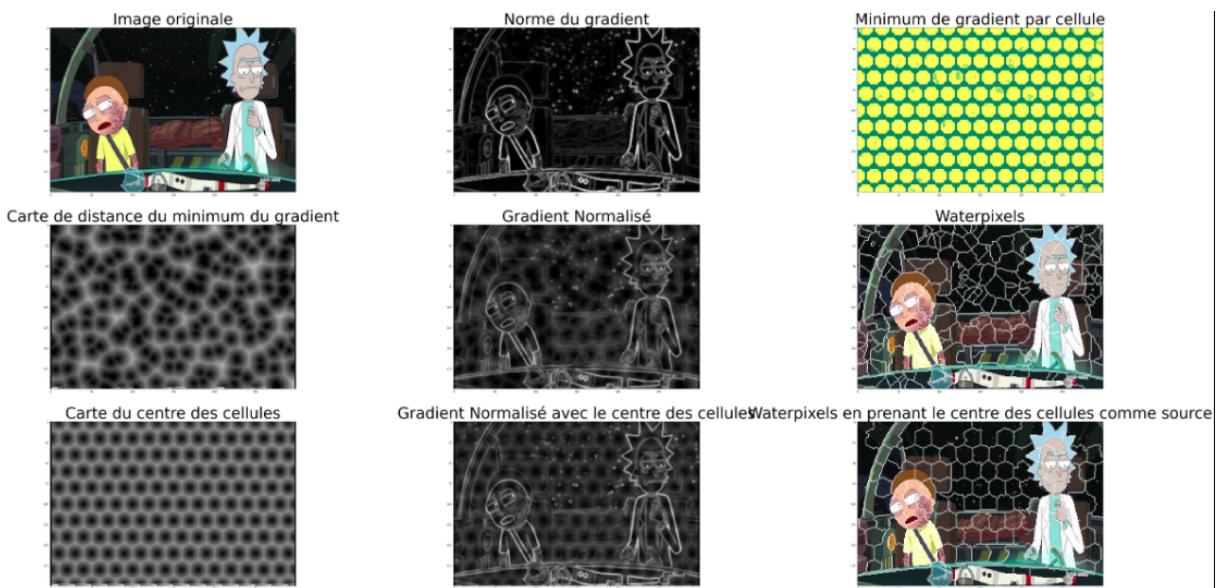


FIGURE 15 – Pour rick.png, on observe le même effet (avec $k=10$)

où les contours sont déjà bien dessinés, le watershed avec marqueurs O_i est très efficace.

Réglage du paramètre k

Nous avons déjà vu précédemment l'influence du paramètre k sur l'image. Mais ce paramètre doit être ajusté pour chaque image à la main, car elles ne présentent pas toutes le même profil morphologique.

L'idée de faire varier le k est claire : trouver une bonne alternative entre avoir des pixels qui dessinent bien les contours et ne pas avoir trop de pixels qui s'occupent de dessiner des détails (comme l'étoile par exemple dans l'image du dessin animé).

Avec l'image du canyon, on observe les choses suivantes :

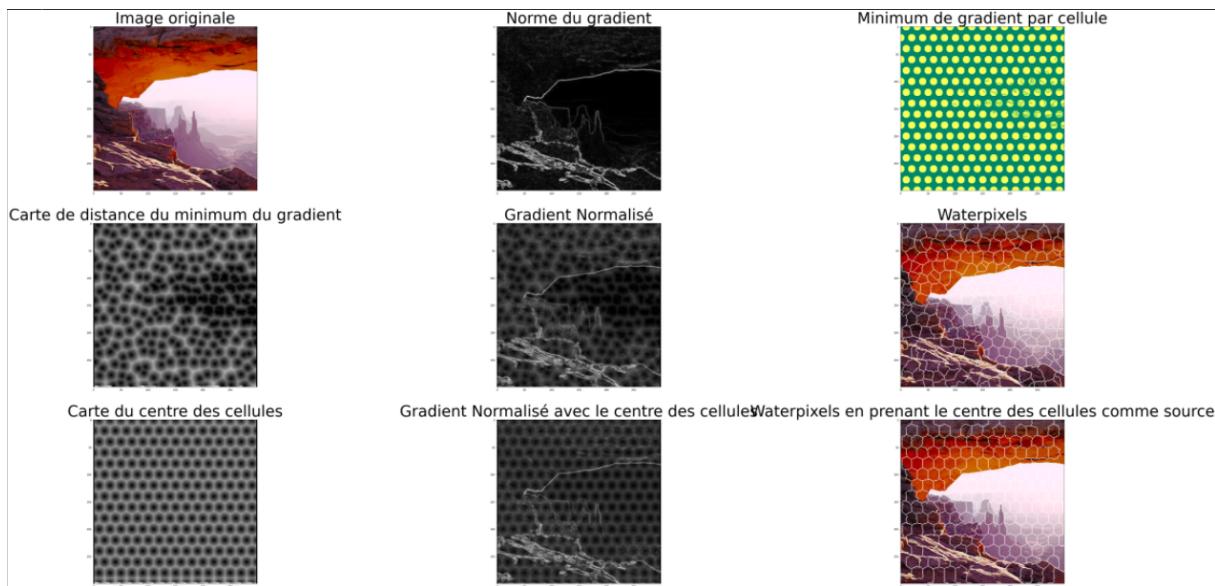


FIGURE 16 – valley.png avec k=15

Certains pixels dessinent mal les contours (pour les marqueurs O_i comme pour les M_i).

On va donc tenter de diminuer le k pour obtenir un résultat plus satisfaisant. On essaie avec k=5 par exemple :

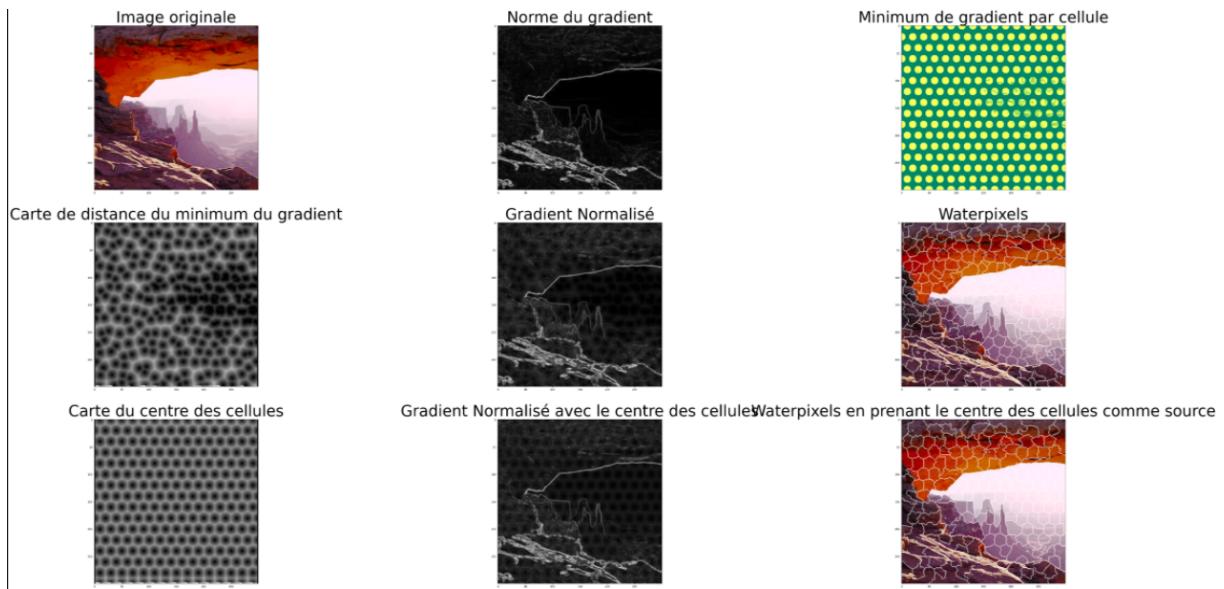


FIGURE 17 – valley.png avec $k=5$

Cette fois-ci les contours sont très bien dessinés par les pixels, pour le watershed avec marqueurs O_i , et M_i . Cependant, on remarque que de tout petits pixels se dessinent, et ce n'est pas ce qui est attendu : le k choisis ici est trop faible.

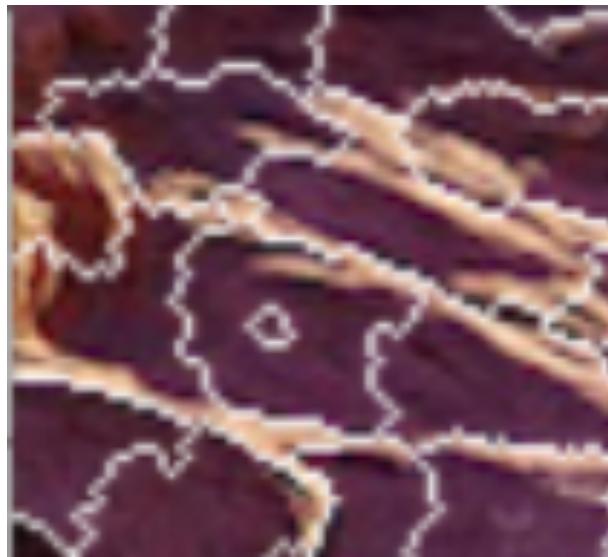


FIGURE 18 – valley.png avec $k=5$: zoom sur un waterpixel trop petit

Comparaison des gradients

Nous l'avons vu, le problème rencontré jusqu'ici est le *ciel*. Le paramètre qui peut permettre de remédier à ce problème est le choix du gradient. En effet, en choisissant un gradient qui permet

de filtrer le *bruit*, on peut obtenir un ciel qui présente un gradient nul partout, en utilisant le gradient de Sobel à la place du gradient morphologique, et en jouant sur le paramètre *seuilnorme*. Le problème est qu'avec le gradient de Sobel, le calcul du gradient normalisé ne donne pas un résultat aussi satisfaisant qu'avec le gradient morphologique : il faut utiliser un *k* beaucoup plus petit (voire un *k* inférieur à un). À cela s'ajoute le fait que les contours sont beaucoup moins bien dessinés avec le gradient de Sobel, comme un peu le voir ci-dessous.

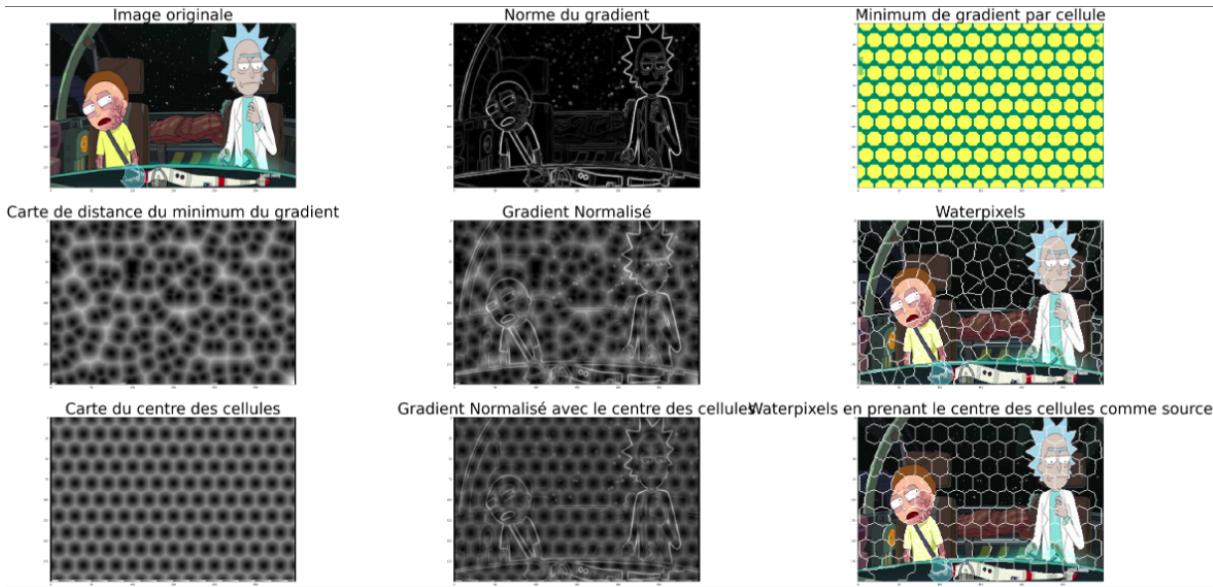


FIGURE 19 – rick.png avec le gradient de sobel ($\sigma = 0.0$ *seuilnorme* = 10) pour $k = 0.5$

Mais cette manipulation permet tout de même d'avoir un *ciel* mieux découpé, comme le montre l'application de watershed avec gradient de sobel à l'image montagne.png que nous avons rencontré précédemment.

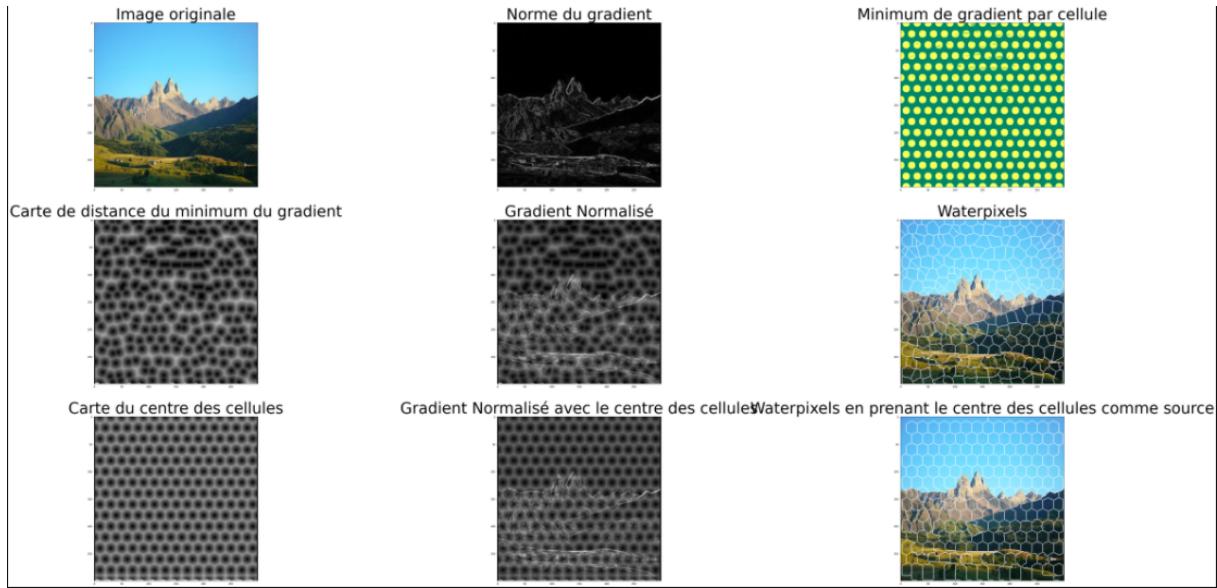


FIGURE 20 – montagne.png avec le gradient de sobel ($\sigma = 0.0$ seuilnorme = 0.001) pour $k = 0.3$

C'est donc au manipulateur de jauger ce qu'il préfère mettre en valeur, notamment en fonction de l'image qu'il utilise (qui présente un *ciel* ou non), afin d'en tirer le meilleur découpage en waterpixels possible.

Enfin, il aurait pu être envisagé d'intégrer dans la fonction *min_grad* un certain seuil, qui permet de juger si la différence entre le maximum et le minimum du gradient, est assez grande pour pouvoir faire mettre un marqueur M_i . Dans le cas contraire on placerait un marqueur O_i , et le ciel serait mieux traité.

Conclusion

Nous avons rencontré beaucoup de problèmes de traitement des images au cours de ce projet, et il a fallu faire des choix pour obtenir un résultat qui convient calcule le mieux les waterpixels.

Nous avons traité simplement des images de petites tailles, car le temps de calcul est bien trop long pour de grandes tailles, mais le résultat obtenu est assez satisfaisant puisque l'algorithme calcule correctement des pixels intelligents qui dessinent des contours.

Enfin, il est possible d'adapter le calcul de ces pixels en fonction de l'image que l'on veut traiter (avec beaucoup de ciel, ou bien avec des zones de fortes turbulences en terme de gradient), en utilisant un gradient différent, ou bien en faisant varier les valeurs de k , ou encore en changeant le paramètre *taille_cellule* (pour avoir de plus ou moins grands pixels).

Bibliographie

[1] WATERPIXELS, Vaïa Machairas, Matthieu Faessel, David Cárdenas-Peña, Théodore Chabardes, Thomas Walter, Etienne Decencière, 7 Oct 2015, <https://hal.archives-ouvertes.fr/hal-01212760>

[2] LES DISTANCES DE CHANFREIN EN ANALYSE D'IMAGES : FONDEMENTS ET APPLICATIONS, Thèse d'Edouard Thiel, soutenue le 21 Septembre 1994, <https://tel.archives-ouvertes.fr/tel-00005113>