Rapport SY32 - Mise en correspondace stéreo

NGUYEN Thi Thu Uyen, LAVISSE Théotime

8 mai 2022

1 Objectif

L'objectif de ce projet a été d'implémenter un algorithme de mise en correspondance stéréo et de le tester sur les images du jeu de données suivant : Cliquez ici pour accéder aux données. Pour évaluer la performance de nos algorithmes, nous devions prendre en compte la qualité (pourcentage de pixels avec une erreur de disparité), la rapidité d'exécution ainsi que le ratio qualité / rapidité. Pour nous aider dans cette évaluation, nous disposions des disparités vérité correspondant aux images à tester, ainsi qu'aux cartes d'occultations pour l'image de gauche.

2 Choix de l'algorithme

Nous avons testé trois algorithmes de mise en correspondance stéréo : le Block Matching, le Semi-Global Block Matching ainsi que la Programmation Dynamique. Sans approfondir le travail de pré et post-traitement, nous souhaitions évaluer ces différents algorithmes selon les critères cités précédemment afin de choisir le plus "efficace" et nous concentrer dessus pour la suite. Après nos premières expérimentations sur ces algorithmes, nous sommes arrivés au résultat suivant sur le jeu de données "Cones" :

Algorithme	Temps d'exécution	Pourcentage de disparités dont l'erreur $>$ 1px	Pourcentage de disparités dont l'erreur $>$ 2px
Semi-Global Block Matching	67 secondes	18.75 %	17.58 %
Block matching	7 secondes	27.96 %	24.79 %
Programmation dynamique	36 secondes	41.13 %	34.71 %

Les résultats sur le jeu de données "Teddy" sont présentés ci-dessous :

Algorithme	Temps d'exécution	Pourcentage de disparités dont l'erreur $>$ 1px	Pourcentage de disparités dont l'erreur $>$à 2px
Semi-Global Block Matching	64 secondes	19.51 %	17.19 %
Block matching	7 secondes	31.67 %	28.20 %
Programmation dynamique	36 secondes	33.40 %	27.30 %

Au vu de ces résultats, on constate que, sur les données testées, l'algorithme présentant les résultats plus précis est clairement le Semi-Global Block Matching. Cependant, son temps d'exécution est égale-

ment le plus lent. Si l'on s'intéresse aux critères de rapidité et de ratio qualité / rapidité, l'algorithme le plus efficace est le Block Matching. Pour cettes raisons, nous avons donc retenu pour la suite de notre travail d'optimisation le Block Matching pour la rapidité, et le Semi-Global Block Matching pour la qualité.

3 Block Matching

3.1 Principe

L'algorithme de Block Matching est une méthode locale qui évalue un pixel à la fois, en ne considérant que les pixels de son voisinage. Dans cet algorithme, un "bloc" désigne une fenêtre n*n de pixels autour d'un pixel central. Son objectif est de trouver pour chaque bloc de l'image de référence, le bloc le plus similaire à celui-ci dans la seconde image, sur la même ligne épipolaire, et d'enregistrer la distance horizontale entre les deux : la disparité.

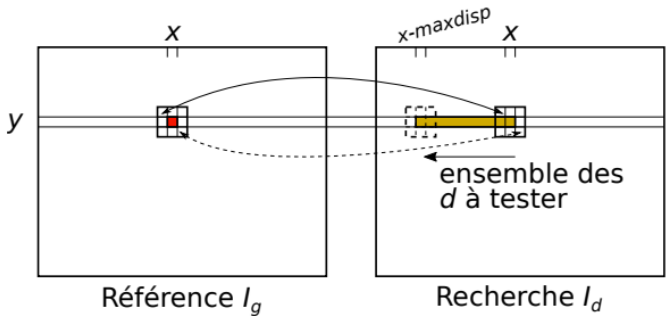

FIGURE 1 – Principe du Block Matching.

Pour la mesure de dissimilarité entre blocs, nous avons testé les méthodes SAD (Sum of Absolute Differences) et SSE (Sum of Squared Errors) :

$$SAD(bloc1, bloc2) = \sum_{(u,v)\in N\times N} |bloc2(u,v) - bloc1(u,v)|$$

$$SSE(bloc1, bloc2) = \sum_{(u,v)\in N\times N} (bloc2(u,v) - bloc1(u,v))^2$$

Les résultats sur le jeu de données "Cones" en utilisant les méthodes SAD et SSE sont présentés ci-dessous :

Mesure de dissimilarité	Erreur moyenne hors occultations	Pourcentage de disparités dont l'erreur > 1px	Pourcentage de disparités dont l'erreur > 2px
SSE	1.59 px	27.96 %	24.79 %
SAD	1.75 px	30.65 %	26.94 %

On constate que les résultats obtenus par SSE sont meilleurs que ceux obtenus par SAD. Nous avons donc retenu le SSE pour la mesure de dissimilarité dans notre code final.

En pratique, on fixe une disparité maximale qui sera le décalage maximal d'une image à l'autre pour un bloc. La carte des disparités que l'on obtient en sortie de l'algorithme correspond, pour chaque pixel de coordonnées (x,y) à la distance en pixels entre le bloc centré autour de ce pixel dans l'image initiale et le bloc le plus similaire dans la seconde.

Pour réduire le temps de calcul et améliorer la robustesse de notre algorithme, nous avions une disparité maximale de 64 et supposions la contrainte d'unicité (les appariements multiples ne sont pas autorisés).

Pour notre implémentation, nous utilisions l'image de gauche comme référence pour le calcul des disparités. Concernant les paramètres de l'algorithme, il nous restait à fixer la taille des blocs.

3.2 Optimisation

3.2.1 Implémentation

La première implémentation de cet algorithme nous a donné des résultats moyens (ceux présentés dans les tableaux précédents), et un temps d'éxécution de 5 minutes et 30 secondes. Cependant, nous avons utilisé les fonctionnalités de la librairie numpy pour grandement accélérer le traitement. Au lieu d'itérer sur toutes les lignes et toutes les colonnes, puis sur toutes les disparités possibles, nous avons utilisé les "stride tricks" de numpy qui sont très utiles pour créer des fenêtres coulissantes sur des tableaux. De plus, nous avons regroupé le calcul des 64 dissimilarités pour chaque bloc en un seul calcul vectoriel. Ces optimisations nous ont permis de réduire le temps d'exécution de l'algorithme de 5 minutes 30 à 7 secondes seulement.

3.2.2 Pré-filtrage

Pour améliorer la qualité de notre algorithme de Block Matching, nous avons essayé d'appliquer des pré et post-filtrages sur notre jeu de données. Nous avons d'abord testé un filtre gaussien de taille 3 x 3 sur nos images de test pour diminuer le bruit avant d'estimer les disparités avec le Block Matching. Les résultats sur le jeu de données "Cones" avec et sans pré-filtrage ont été les suivants :

	Erreur moyenne hors occultations	Pourcentage de disparités dont l'erreur $>$ 1px	Pourcentage de disparités dont l'erreur $>$ 2px
Sans pré-filtrage gaussien	1.59 px	27.96 %	24.79 %
Avec pré-filtrage gaussien	4.22 px	72.74 %	53.75 %

Et les résultats sur le jeu de donnée "Teddy" avec et sans pré-filtrage :

	Erreur moyenne hors occultations	Pourcentage de disparités dont l'erreur $>$ 1px	Pourcentage de disparités dont l'erreur $>$ 2px
Sans pré-filtrage gaussien	2.32 px	31.67 %	28.20 %
Avec pré-filtrage gaussien	3.99 px	55.80 %	42.48 %

On remarque qu'après avoir débruité nos images avec le filtre gaussien, les qualités obtenues sont diminuées. En effet, les images sont déjà rectifiées de telle sorte que les lignes de même hauteur soient les lignes épipolaires conjuguées. En utilisant le filtre gaussien, on a modifié cette rectification et donc le Block Matching ne peut pas bien fonctionner. A cause de cette raison, nous n'avons pas appliqué le pré-filtrage dans l'optimisation.

3.2.3 Post-filtrage et Choix des paramètres

Comme précisé précédemment, le paramètre sur lequel nous pouvions jouer pour l'algorithme de Block Matching était la taille des blocs. Pour trouver la meilleure taille de bloc, nous avons utilisé un algorithme de recherche exhaustive. Celui-ci consiste à étudier toutes les tailles possibles pour choisir la meilleure parmi celles-ci. Nous avons utilisé cet algorithme pour les tailles de blocs : 3, 5, 7, 9 et 11 qui sont les tailles communément utilisées, et qui nous donnaient à première vue les meilleurs résultats.

On constate que pour des tailles de voisinage réduites, la carte de disparité contient plus de détails mais aussi plus de bruit impulsionnel.

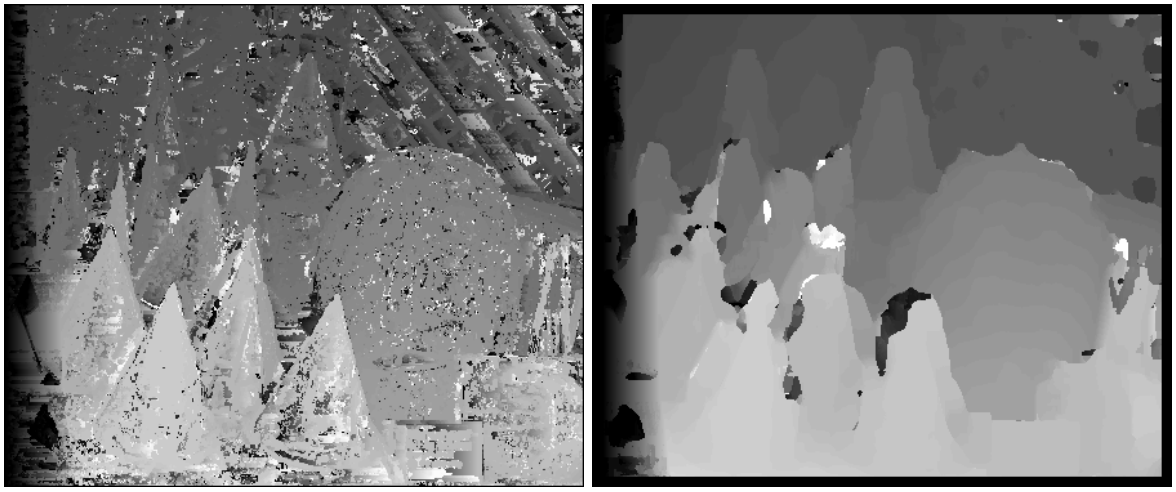

FIGURE 2 – Block Matching sur les images "Cones" avec les tailles du fenêtre 3 et 17.

Pour cette raison, nous nous sommes également intéressés à l'utilisation d'un filtre médian en post-filtrage. En effet, il pouvait permettre de réduire le bruit, en particulier l'effet "poivre et sel" dans notre carte de dissimilarités finale. Pour celui-ci, nous avons également utilisé la recherche exhaustive pour sélectionner sa taille parmi : 0, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21. La taille 0 correspondant à la non utilisation du filtre. Nous avons obtenu les résultats suivants :

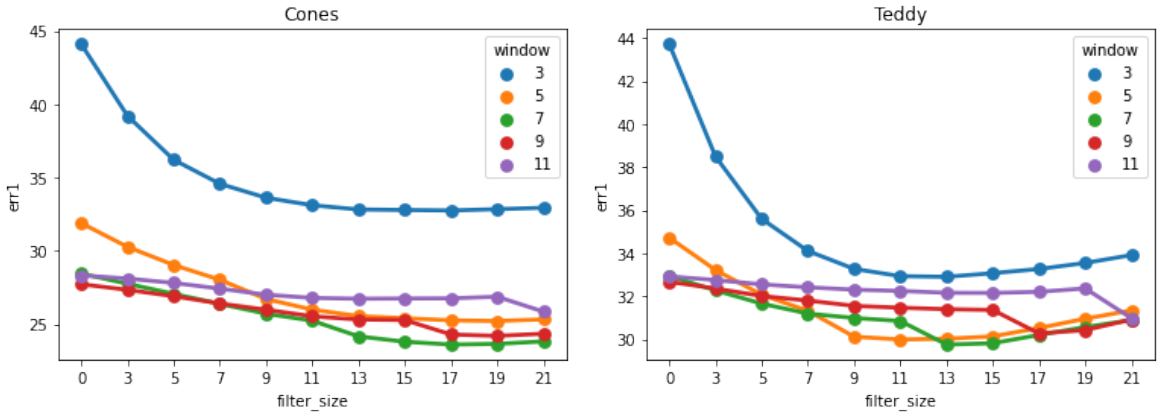

FIGURE 3 – Graphiques des taux de disparités dont l'erreur est supérieure à 1px selon la taille du filtre, la taille du bloc ainsi que l'image.

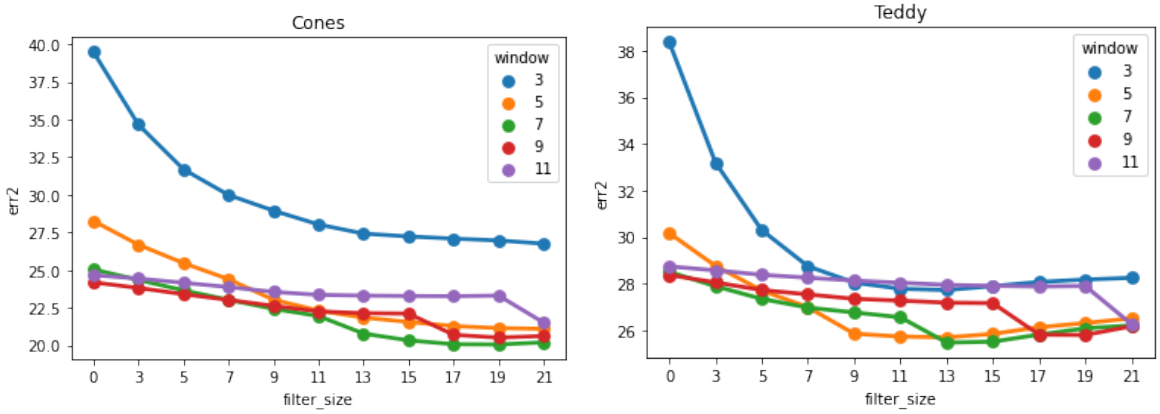

FIGURE 4 – Graphiques des taux de disparités dont l'erreur est supérieure à 2px selon la taille du filtre, la taille du bloc ainsi que l'image.

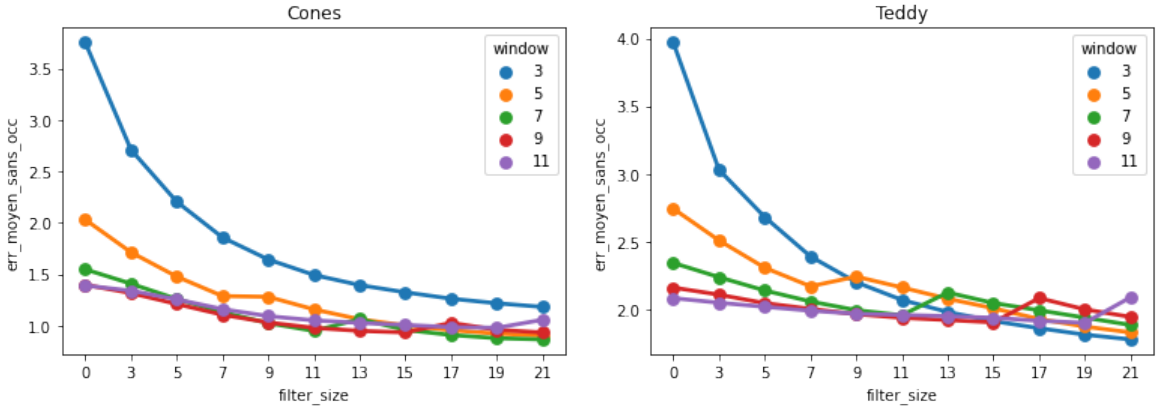

FIGURE 5 – Graphiques des erreurs moyenne des disparités estimées selon la taille du filtre, la taille du bloc ainsi que l'image.

Au vu de ces résultats, nous avons pu constater que le post-filtrage avec un filtre médian donnait de très bon résultats et faisait baisser toutes les mesures d'erreurs dont nous disposions. Nous avons donc choisi de l'utiliser dans notre code final. Concernant le choix des paramètres, nous avons sélectionné la taille de fenêtre 7, et la taille du filtre à 17, combinaison la plus robuste.

3.2.4 Post-traitement

Remplissage des bords. L'algorithme de Block Matching traitant des blocs, les pixels des bords de l'image de référence n'ont pas pu être pris en compte dans l'algorithme. Ainsi, une bande d'une largeur correspondant à la moitié de la taille d'un bloc restait vide dans la carte des disparités en sortie. Nous avons choisi de la remplir avec les valeurs des pixels calculés les plus proches. Cette petite opération a permis de réduire l'erreur de 0.5 % en moyenne.

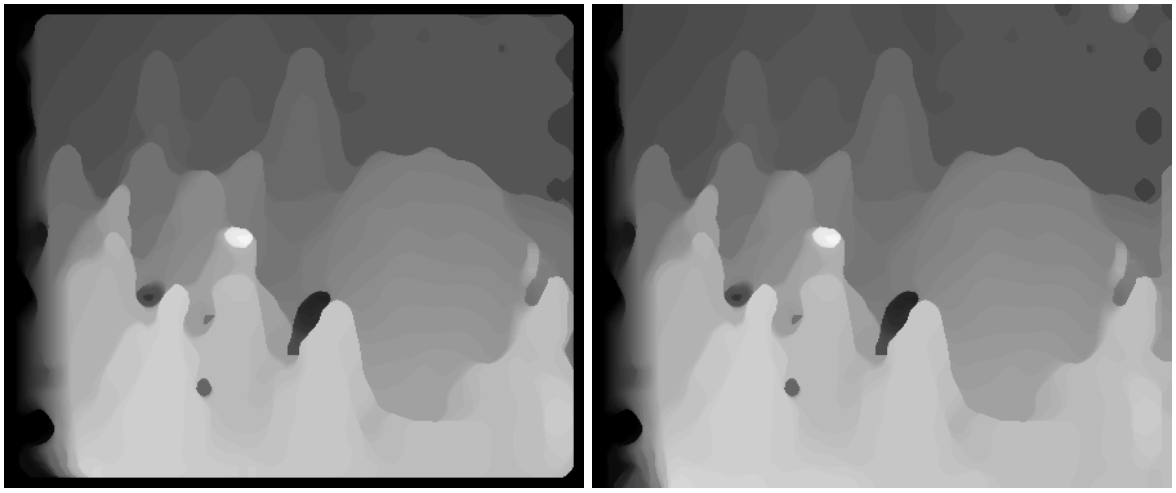

FIGURE 6 – Cartes de disparités sur les images "Cones" avant et après le remplissage des bords.

Seuillage des régions occultées. Une des grandes limites du Block Matching est erreurs aux occultations. Donc, nous nous sommes penchés sur l'amélioration des régions occultées. Ces zones correspondent à des parties de la scène qui étaient visibles sur l'image initiale mais qui ne le sont pas sur la seconde. Elles doivent être codées par des zéros sur la carte des disparités, et sont prises en compte dans le calcul des erreurs (par comparaison avec les cartes d'occultation).

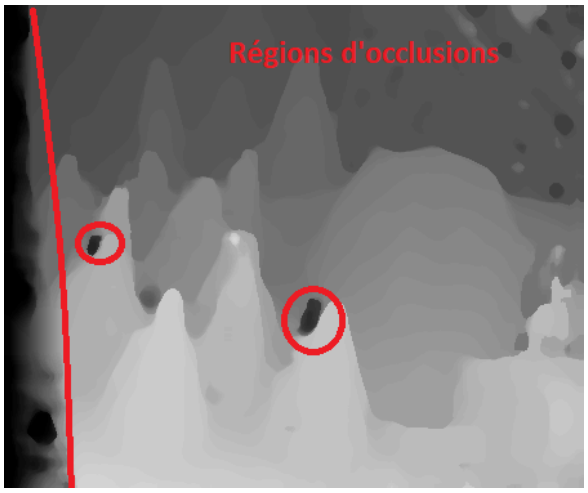

FIGURE 7 – Régions occultées dans la carte de disparités "Cones" obtenue par Block Matching

Nous remarquons que les régions occultées sont les régions plus sombres dans la carte des disparités. Pour cette raison, nous avons souhaité appliquer un seuillage sur notre image de sortie pour que les zones qui sont sombres, sans être pour autant codées par un zéro, soient remplacées par des zéros pour limiter les erreurs liées aux zones occultées. Pour choisir ce seuil, nous avons encore une fois procédé par recherche exhaustive et avons obtenu les résultats suivants :

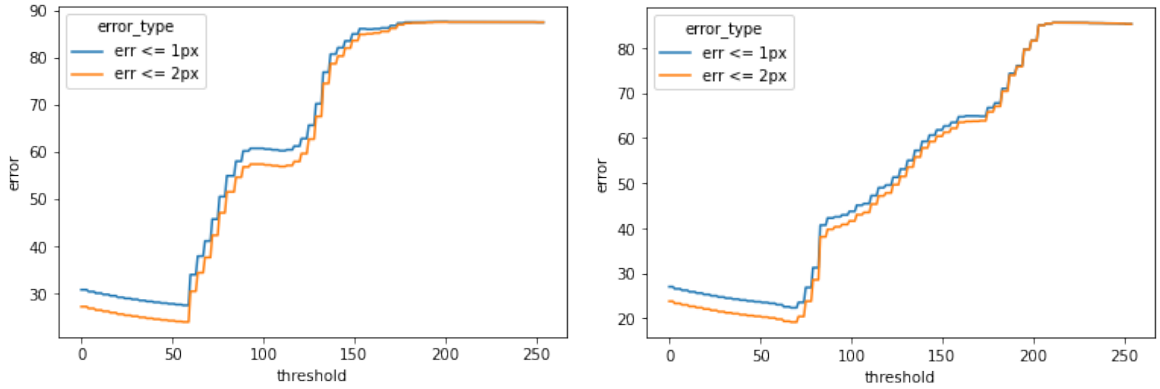

FIGURE 8 – Graphiques des taux de disparités dont l'erreur est supérieure à 1 ou 2px selon le seuil appliqué, pour le jeu de données "Cones" à gauche et "Teddy" à droite

On constate pour le jeu de données "Cones" que l'erreur baisse tant que le seuil augmente jusqu'à 59px, puis elle est croissante. Pour le jeu de données "Teddy", le constat est le même jusqu'au seuil de valeur 70px. Ainsi, nous choisissons pour notre algorithme d'utiliser un seuil de 59.

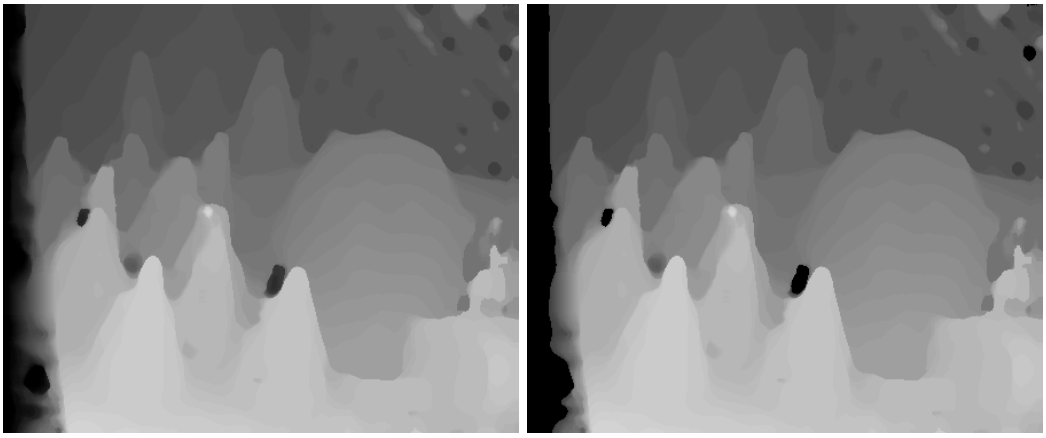

FIGURE 9 – Cartes de disparités pour le jeu de données "Cones" avant et après avoir appliqué le seuillage 59px.

Cette opération a permis de réduire l'erreur de 4 % en moyenne pour les jeux de données "Cones" et "Teddy".

3.3 Visualisation graphique du Block Matching avec optimisations

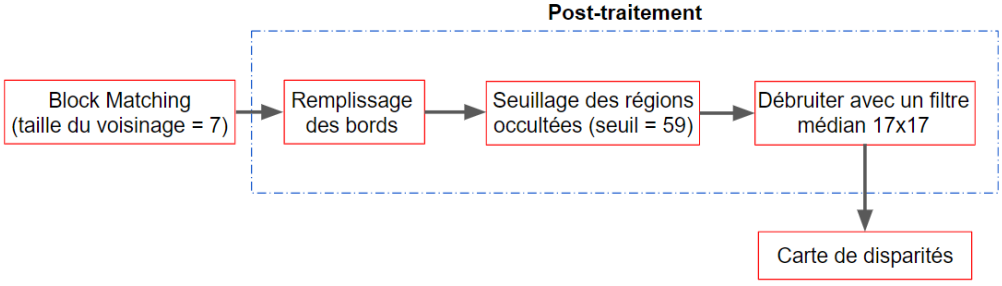

FIGURE 10 – L'algorithme de Block Matching optimisé étape par étape.

4 Semi-Global Block Matching

4.1 Principe

Les méthodes globales de mise en correspondance minimisent le coût global et sont généralement plus précises et robustes que les méthodes locales. L'algorithme de Semi-Global Matching utilise les informations des pixels du voisinage dans plusieurs directions pour calculer la disparité d'un pixel. Cependant, cette analyse multidirectionnelle entraîne de nombreux calculs. Ainsi, au lieu d'utiliser l'image entière, la disparité d'un pixel peut être calculée en considérant un bloc de pixels pour faciliter le calcul. L'algorithme Semi-Global Block Matching (SGBM) utilise une correspondance de coût basée sur les blocs qui est lissée par des informations de cheminement provenant de plusieurs directions.

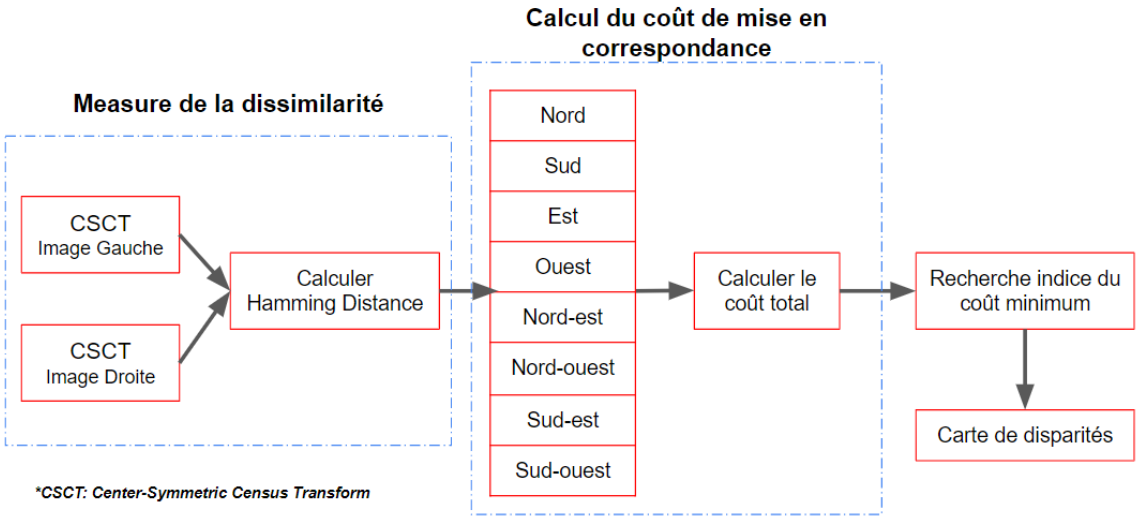

FIGURE 11 – L'algorithme de SGBM étape par étape.

L'algorithme de SGBM comprend 2 modules principaux : mesure de la dissimilarité et calcul du coût de mise en correspondance.

Mesure de la dissimilarité : Ce module comprend 2 étape : calcul CSCT (Center-Symmetric Census Transform) pour les images de gauche et de droite et puis calcul de la distance de Hamming.

Pour le calcul CSCT, un "bloc" désigne une fenêtre n*m de pixels autour d'un pixel central. Il consiste à transformer chaque bloc des images de gauche et de droits en un nouveau bloc comprenant les binaires 1 et 0. Pour ce faire, on compare les valeurs de chaque pixel avec la valeur du pixel central. Si la valeur est plus grande que la valeur du pixel central, on mettra 1 dans le bloc de census, sinon on y mettra 0. La taille du bloc est un paramètre à fixer.

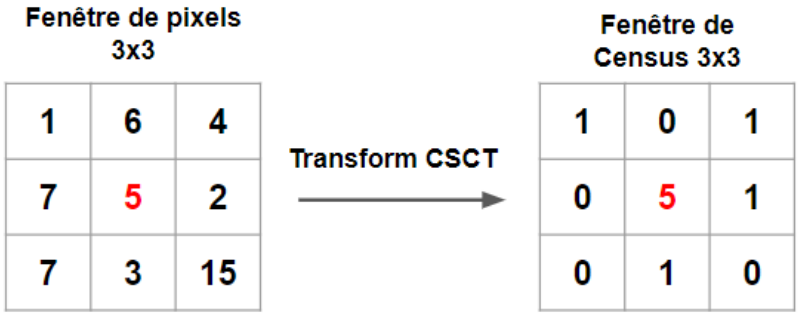

FIGURE 12 – Exemple de Center-Symmetric Census Transform CSCT.

Ensuite, la distance de Hamming est calculée pour mesurer la dissimilarité entre les blocs. La distance de Hamming entre un bloc B1 de l'image gauche et un bloc B2 de l'image droit est le pixelwise XOR entre $CSCT(B1)$ et $CSCT(B2)$. On construit une matrice du coût $D(p,d)$ contenant la distance de Hamming entre le pixel p de l'image de gauche et le pixel $p-d$ de l'image de droite.

Calcul du coût de mise en correspondance : On utilise la matrice $D(p,d)$ et un terme de régularisation binaire $R(d_p, d_q)$ avec :

$$R(d_p, d_q) = \begin{cases} 0 & \text{si } d_p = d_q \\ P1 & \text{si } |d_p - d_q| = 1 \\ P2 & \text{si } |d_p - d_q| > 1 \end{cases}$$

où $P1, P2$ sont les paramètres de pénalité à fixer avec $P1 < P2$. Le calcul du coût de mise en correspondance est inspiré par la méthode de Programmation Dynamique. Cependant, pour le SGBM, on calcule les coûts dans 8 directions comme ci-dessous :

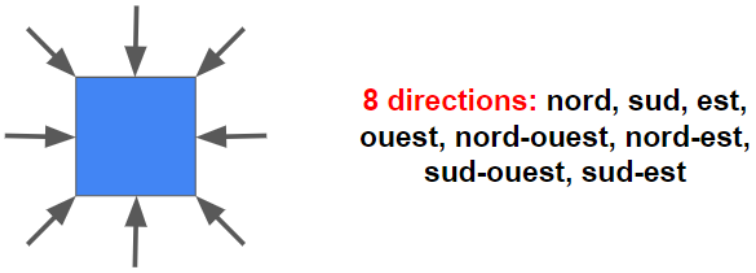

FIGURE 13 – Directions de calcul le coût de mise en correspondance.

Pour chaque direction, on construit la matrice $L_r(p,d)$ contenant les coûts pour atteindre le pixel p avec un niveau de disparité d dans la direction r avec :

$$L_r(p,d) = D(p,d) + min\{L_r(p-r,d), L_r(P-r,d-1) + P_1, L_r(P-r,i) + P_2\} - minL_r(p-r,k)$$

où

$L_r(p-r,d-1)$ *le coût précédent du pixel p avec la disparité d-1 dans la direction r.*

$L_r(p-r, d+1)$ *le coût précédent du pixel p avec la disparité d+1 dans la direction r.*

$minL_r(p-r, k)$ *le coût minimum de la direction r pour l'itération précédente.*

Enfin, le coût total de mise en correspondance $S(p, d)$ est la somme totale de $L_r(p, d)$ dans toutes les directions. Donc, pour chaque pixel p, la disparité finale est la disparité d qui donne la valeur minimum pour $S(p, d)$.

4.2 Optimisation

4.2.1 Réduction des directions de recherche

Pour l'algorithme de Semi-Global Block Matching, la précision du résultat de la disparité s'améliore avec l'augmentation du nombre de directions, cependant, le temps d'exécution augmente également. Pour améliorer la rapidité de cet algorithme, on cherche à réduire le nombre de directions. Nous avons choisi pour notre code final de calculer le coût total de mise en correspondance dans 4 directions : nord, sud, est et ouest.

4.2.2 Choix des paramètres et Post-traitement

Pour le choix des paramètres, nous avons utilisé la recherche exhaustive similaire aux optimisations de Block Matching pour trouver les meilleures valeurs pour la taille du bloc et les pénalités P1, P2. Les valeurs utilisées dans notre code final sont donc : P1 = 10, P2 = 30, taille du bloc : 5x5.

Les opérations de post-traitement sont similaires à l'algorithme de Block Matching. On applique d'abord un seuillage pour améliorer les régions occultées (seuil = 55) et puis un filtre médian de taille 5x5 pour éliminer.

4.2.3 Visualisation graphique du SGBM avec optimisations

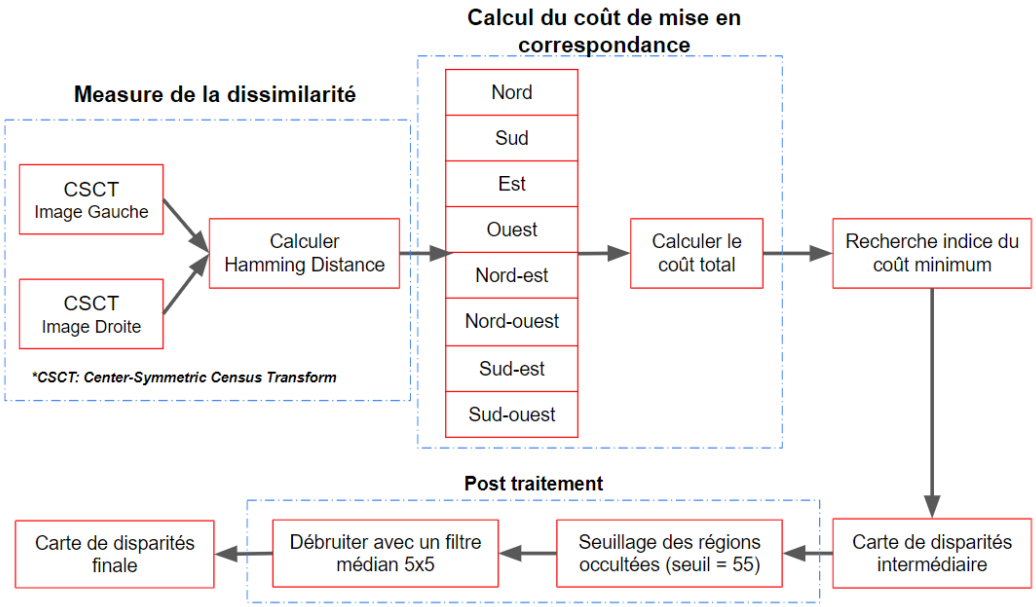

FIGURE 14 – L'algorithme de SGBM optimisé étape par étape.

5 Résultats obtenus et Conclusions

Après nos optimisations sur les algorithmes de Block Matching et Semi-Global Block Matching, nous sommes arrivés aux résultats suivants sur les jeux de données "Cones" et "Teddy" :

	Temps d'exécution	Erreur moyenne hors occultations	% de disparités dont l'erreur > 1px	% de disparités dont l'erreur > 2px
Teddy SGBM	22 secondes	0.75 px	14.36 %	11.63 %
Teddy BM	7 secondes	1.39 px	25.08 %	21.09 %
Cones SGBM	21 secondes	0.63 px	12.88 %	11.91 %
Cones BM	7 secondes	0.87 px	18.77 %	15.71 %

Au vu de ces résultats, on constate que l'algorithme de Semi-Global Block Matching nous donne les résultats les plus précis et les plus robustes. Cependant, en termes de rapidité et de ratio qualité/rapidité, l'algorithme le plus efficace est le Block Matching.

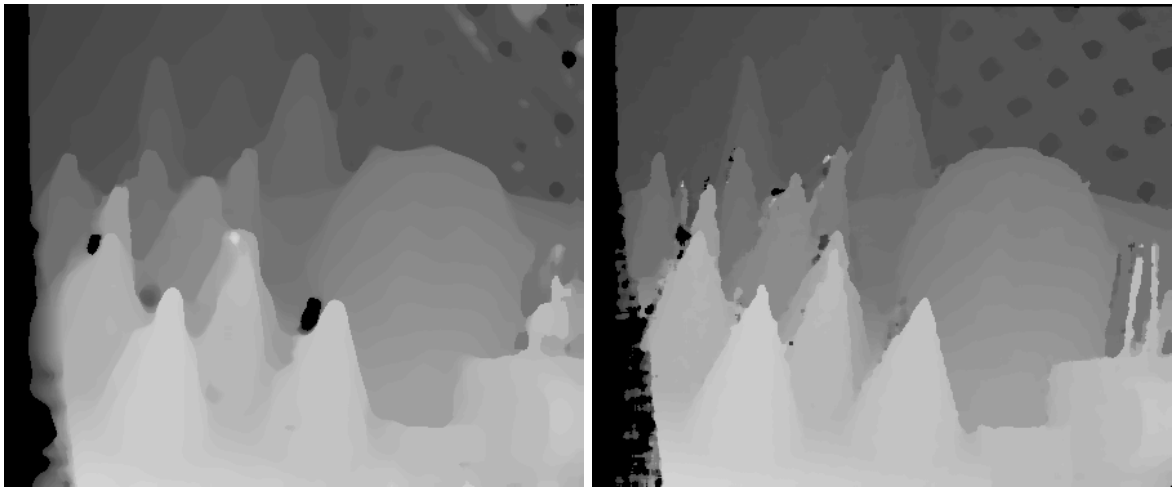

FIGURE 15 – Cartes de disparités finales pour le jeu de données "Cones" obtenues par le Block Matching (gauche) et le Semi-Global Block Matching (droit).

Selon les cartes de disparités obtenues sur le jeu de données "Cones" au dessus, on remarque que les contours de objets sont plus clairs pour le Semi-Global Block Matching.