Stéréo

IFT6145 : Vision 3D - TP #4

Jérémie Dumas

25 avril 2011

Université de Montréal

Table des matières

1 Les algorithmes de stéréo

1.1 Présentation du problème

Dans son énoncé le plus général, le problème de la stéréo consiste à retrouver une correspondance dense entre deux images prises dans des vues ayant subies une translation pure (même paramètres internes de caméra donc). On s'intéresse ici au cas particulier des images rectifiées (lignes épipolaires parallèles), et sans distorsion radiale.

Concrètement cela signifie que les correspondances se font sur une même ligne horizontale, et que la profondeur d'un pixel dans l'espace d'une caméra est inversement proportionnel à sa disparité entre les deux images. Le problème peut alors se formuler de la manière suivante :

Définition 1.1 (Recherche Stéréo)**.** Soient I_1, I_2 deux images rectifiées de taille $h \times w$. Pour chaque ligne $0 \leqslant y < h$, trouver une relation $\sim_y$ telle que $x_1 \sim_y x_2$ ssi les pixels $I_1(x_1, y)$ et $I_2(x_2, y)$ sont en correspondance.

Remarque. On considère ici un mouvement gauche-droite, donc si $x_1 \sim_y x_2$, alors $x_1 \geqslant x_2$.

Dans les algorithmes qui suivent, on cherchera $\sim_y$ sous forme de fonction (injective pour l'algo. 2), que l'on pourra représenter avec une carte de disparité : $d_y(x_1) = x_1 - x_2$ si $x_1 \sim_y x_2$, et $d(x_1) = 0$ sinon (occultation) — le cas $x_1 = x_2$ ne se produisant pas ici.

Les disparités considérées seront comprises entre deux valeurs $0 \leqslant d \leqslant d_{max}$.

1.2 Fonction de coût et lissage

Dans la suite de cette section, on utilise des heuristiques qui dépendent du choix d'une fonction de coût f (pour mettre en correspondance deux pixels), ainsi que d'un terme de lissage g (pour pénaliser des voisins ayant deux disparités trop éloignées).

La fonction de coût sera abrégée en $f_y(x_1, x_2)$. On peut choisir des pénalités classiques comme SSD ou SAD, mais aussi d'autres fonctions de coût comme Birchfield-Tomasi. La fonction de lissage n'est utilisée que pour le dernier algorithme, et sera notée $g_y(d, d')$, où d et d' désignent les disparités associées à deux pixels voisins x_1 et $x_1 + 1$.

1.3 Algo. 1 : *Winner Takes All*

L'idée est simple : à chaque pixel de I_1 on associe le pixel de I_2 qui minimise la fonction de coût $f_y(x_1, x_2)$. Ici pas de lissage entre les disparités, pas de contraintes d'ordre ni d'unicité.

Programme 1 Winner Takes All

Entrée : Deux images rectifiées I_1 (gauche) et I_2 (droite) de taille $h \times w$.
Sortie : Une carte de disparité $D[x, y]$ pour I_1.

```
Pour y = 0 à h − 1 faire
    Pour x = 0 à w − 1 faire
        D[x, y] ← arg min_{d∈⟦0,d_max⟧}(f_y(x, x − d)) // Coût de match
    Fin Pour
Fin Pour
Retourner D
```

Complexité : $\mathcal{O}(hw \cdot d_{max})$

1.4 Algo. 2 : Programmation dynamique

Considérons ici une ligne y de l'image. On raisonne ici sur les lignes partielles de 0 à x_1 pour I_1, et de 0 à x_2 pour I_2, en imposant la contrainte d'ordre et d'unicité sur les matchs obtenus. On cherche à minimiser la somme suivante pour une relation $\sim$ donnée :

$$S(x_1, x_2) = \sum_{\substack{i \leqslant x_1, j \leqslant x_2 \\ i \sim j}} f_y(i, j) + K \cdot N$$

Où $K > 0$ est une constante de pénalité pour les pixels en occultation, et où N est le nombre de pixel n'ayant pas de match : $|\{i, \nexists k, i \sim k\}| + |\{j, \nexists k, k \sim j\}|$.

Notons alors $O(x_1, x_2)$ le coût minimal obtenu pour la meilleure correspondance partielle $\sim_{opt}$ jusqu'à x_1, x_2. Plusieurs cas sont possibles :

$$O(x_1, x_2) = \begin{cases} O(x_1, x_2 - 1) + K & \text{si } x_1 \sim_{opt} j \text{ où } j < x_2 \ (x_2 \text{ occulté}) \\ O(x_1 - 1, x_2 - 1) + f_y(x_1, x_2) & \text{si } x_1 \sim_{opt} x_2 \\ O(x_1 - 1, x_2) + K & \text{si } i \sim_{opt} x_2 \text{ où } i < x_1 \ (x_1 \text{ occulté}) \end{cases} \tag{1}$$

Comme le mouvement est gauche-droite, on ajoute également la contrainte $x_1 \sim x_2$ seulement si $x_1 \geqslant x_2$. En pratique on imposera donc $f_y(x_1, x_2) = +\infty$ si $x_1 < x_2$. Par rapport à l'équation de récurrence (1), cela veut dire que l'on peut ne pas considérer les entrées pour lequelles $x_1 < x_2$ (car il n'y aura pas de mouvement diagonal dans la partie "supérieure droite"). Voir la figure 1 pour une illustration.

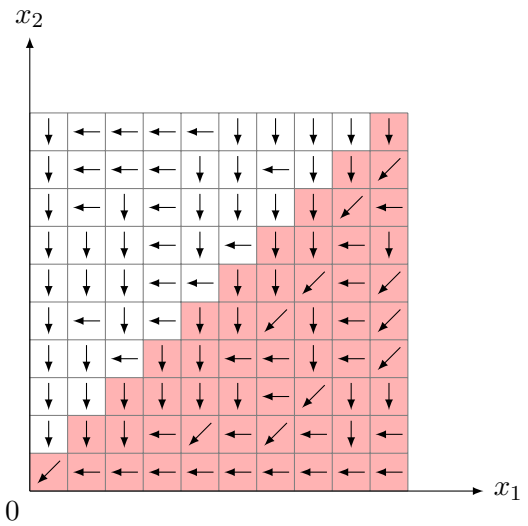


FIGURE 1: Un exemple de prédécesseurs possibles pour la récursion 1

Sur la figure 1, les mouvements au dessus de la diagonale peuvent être omis car ils ne donneront jamais un chemin minimum (il y aura toujours un chemin meilleur qui passe "sous" la diagonale). En considérant enfin la contrainte du mouvement gauche-droite, les conditions aux limites peuvent s'écrire :

$$\begin{aligned}
O(0,0) &= f_y(0,0) \\
O(x_1,0) &= \min(f_y(x_1,0) + K(x_2-1), O(x_1-1,0)+K) \\
O(0,x_2) &= O(0,x_2-1)+K
\end{aligned}$$

Complexité : $\mathcal{O}(hw^2)$

1.5 Algo. 3 : Optimisation de scanlines

Pour ce dernier algo', on raisonne directement sur les disparités associées à chaque pixel. Soit y une ligne de l'image. Considérons la ligne partielle de 0 à x sur I_1 à laquelle on a associé les disparités $d[0], d[1], \ldots, d[x]$. On cherche à minimiser la somme :

$$S(x) = \sum_{0 \leqslant i \leqslant x} f_y(i, i-d[i]) + \lambda \sum_{1 \leqslant i \leqslant x} g_y(d[i-1], d[i]) \tag{2}$$

Notons donc $O(x,d)$ la valeur minimal possible pour cette somme en associant la disparité d au dernier pixel (i.e. $d[x]=d$). On peut exprimer les premiers termes de la somme (2) pour $i < x$ comme la somme optimale obtenue sur la ligne partielle de 0 à $x-1$, en associant à $x-1$ une certaine valeure d'. Cela se traduit donc par :

$$
\begin{aligned}
O(x,d) &= O(x-1,d') + f_y(x,x-d) + \lambda g_y(d',d) \\
&= f_y(x,x-d) + \min_k (O(x-1,k) + \lambda g_y(k,d))
\end{aligned}
$$

Les conditions aux limites ici sont encore plus simples qu'avant, car on a simplement $O(0,d) = f_y(0,d)$. On rappelle aussi que le mouvement étant gauche-droite, (2) nous indique que $O(x,d) = +\infty$ pour $x < d$.

Complexité : $\mathcal{O}(hw \cdot d_{max}^2)$

2 Résultats expérimentaux

2.1 Introduction

Nous fournissons plusieurs scripts qui permettent de tester les 3 algorithmes discutés sur les datasets de Meddleburry [SS02]. L'erreur calculée est le pourcentage de pixels différent du *grountruth* de plus de 1 pixel (mais on peut changer ce seuil facilement).

Le pourcentage est calculé sur les pixels auxquels on a attribué une disparité, donc il faut vérifier qu'un résultat avec une faible erreur ne correspond pas à une image entièrement noire. En effet avec DP par exemple, il peut arriver qu'un ligne soit entièrement mise en occultation, si jamais la pénalité K est trop faible. Il y a aussi du streaking évidemment, et certains artefacts peuvent apparaître si on met des pénalités trop élevées ou trop faibles dans les algo' DP et SO.

2.2 Influence de la méthode choisie

Globalement, on peut voir que les trois méthodes donnent de très mauvais résultats sur Art, mais ne s'en tirent pas trop mal pour le reste. Sur les variantes essayées, nous avons pu constater par exemple que l'algo 3 de *Scanline Optimization*, avec pour fonction de coût f celle de Birchfield-Tomasi, et comme fonction de lissage g le modèle de Potts (Teddy et Venus) ou la SSD (Cônes et Teddy) produit des résultats assez satisfaisants. Bien sûr la recherche n'a pas été exhaustive, mais elle cela donne déjà une idée générale.

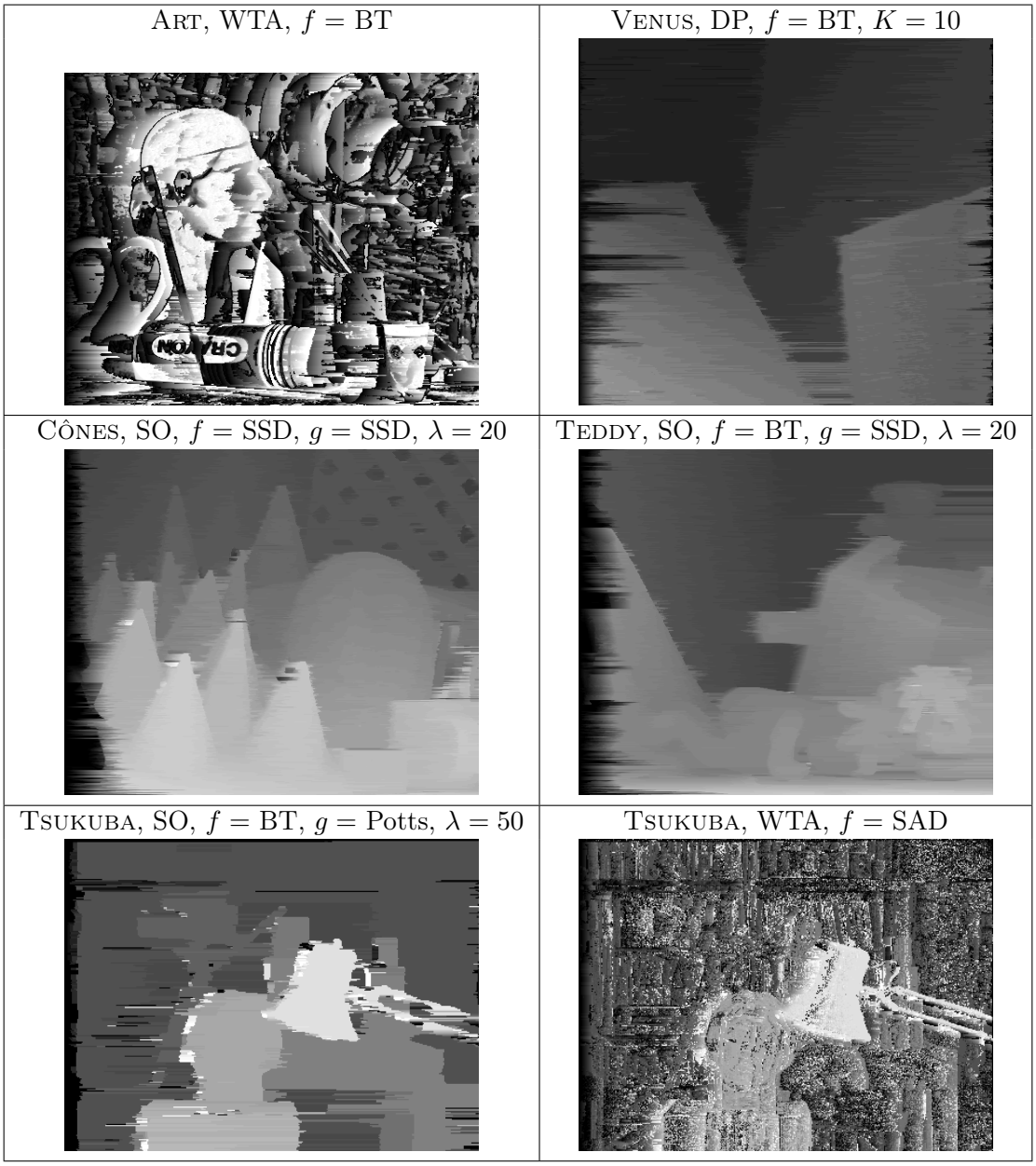


Figure 2: Exemple de résultats obtenus

Sur les différences notables entre les trois algorithmes, on peut dire sans surprise que WTA est assez mauvais (voire figure 2) beaucoup de bruit dans les zones uniforme ou résultat plus *smooth* aurait été appréciable. DP et SO ne sont pas sans problèmes non plus, car si l'optimisation semi-globale au sein d'une ligne se fait bien, les résultats entre les lignes ne sont aucunement corrélés entre eux, ce qui produit un *streaking* assez visible.

Entre DP et SO, les deux peuvent donner des résultats assez similaires (voir figure 3).

On notera pourtant des zones marquées en occultation sur la fin pour DP, ainsi qu'un démarrage plus *smooth* sur la bande de gauche sur le dataset Cônes.

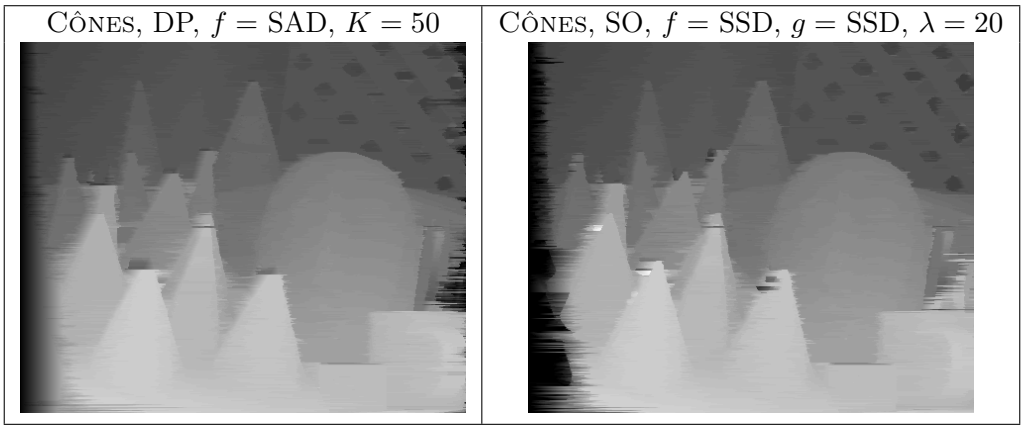


Figure 3: Comparaison entre DP et SO

2.3 Influence de la fonction de coût

Les fonctions de coût SSD et SAD donnent des résultats à peu près identiques, sans grande surprise. On peut tronquer pour en faire des versions robustes, mais dépendamment de la pénalité (K ou λ) que l'on choisit cela n'aidera pas beaucoup : un outlier restera un outlier (surtout avec un seuil de 1 pixel sur la différence autorisée avec le groundtruth).

La fonction de Birchfield-Tomasi donnent des résultats assez similaires à SSD et SAD, comme on peut le voir sur la figure 4. Les performances sont d'ailleurs meilleures avec BT qu'avec SSD ou SAD. Notons quand même le résultat obtenu avec l'algo WTA, qui ne fait pas vraiment de sens (les ambiguïtés sont comme amplifiées).

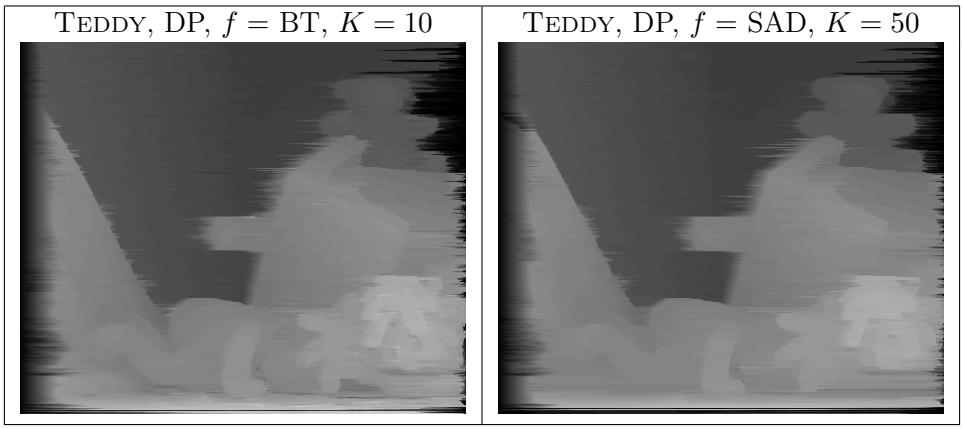


Figure 4: Comparaison entre BT et SAD

Enfin la fonction Census donne des résultats complètement aléatoires avec WTA, mais fait plus de sens avec DP ou SO. Cependant vu le choix du voisinage qui a été fait (3×3), la fonction de coût ne varie que de 0 à 8, ce qui fait assez peu pour bien désambiguer deux pixels. Cela explique sans doute pourquoi les résultats obtenus avec Census ne sont pas meilleurs qu'avec les autres fonctions de coût.

2.4 Influence de la fonction de lissage

Là encore assez peu de différence entre SSD, SAD ou leur version robuste au niveau du lissage. Les résultats sont conformes à nos attentes en terme de performances. On notera que quand le lissage est trop fort (constante λ trop élevée), on se retrouve avec des lignes qui présentent peu de variations de disparités, et donc encore plus de *streaking*. On peut aussi se retrouver avec quelques lignes totalement noires, mais leur origine n'est pas très claire.

Enfin lorsqu'on utilise le modèle de Potts pour le lissage, là on observe une différence notoire sur la carte de disparité. On observe entre autre (voir figure 5) :

- Moins de streaking, car les discontinuités de disparité ne sont pas pénalisées en fonction de leur amplitude (et donc une zone avec une même disparité ne "s'étire" pas sur le reste de la ligne).
- Un résultat meilleur dans les zones continues, car les éléments fronto-parallèles se retrouvent avec la même disparité exactement.
- Des artefacts un peu partout sur les discontinuités, car dès que deux pixels voisins ont des disparités différentes, celle-ci peut varier énormément (d'où les quelques passages noirs).

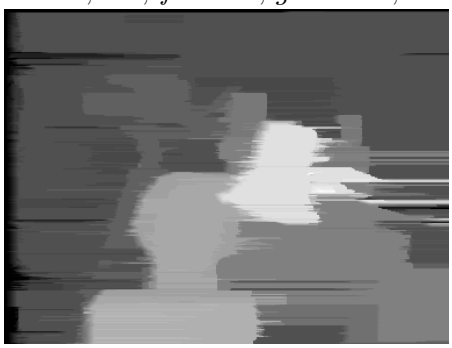
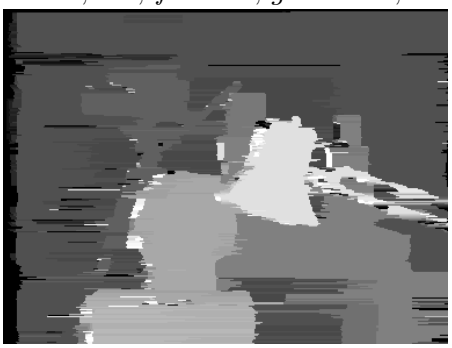
Tsukuba, SO, f = BT, g = SAD, $\lambda = 50$	Tsukuba, SO, f = BT, g = Potts, $\lambda = 50$
	

Figure 5: Comparaison entre SAD et Potts

Conclusion

Des méthodes semi-globales efficaces mais qui souffrent encore de pas mal d'artefact. L'algorithme naïf qui est souvent dans les choux. Finalement la fonction de coût n'est pas le facteur principal qui détermine la qualité du résultat obtenu. Quant au lissage le modèle de Potts présente quelques inconvénients visuels. Cependant les résultats obtenus dans les zones continues, et la possibilité de l'implanter en $\mathcal{O}(hw \cdot d_{max})$ en fait une alternative très intéressante.

On finira pas signaler qu'il existe d'autre méthodes semi-globales et globales qui fonctionnent mieux que ça. Elles ont été vues en cours, mais n'ont pas été implantées dans ce TP, car il y a beaucoup de variante. Et étonnamment certaines fonctions semi-globales donnent de meilleurs résultats, et plus rapidement, que des méthodes globales développées plus tôt.

Références

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47 :7–42, April 2002.