Université Lille1
Sciences et Technologies

Université Lille 1

Stéréovision dense

M2 IVI - M3DA

12/10/2013

par

Bruno Ordozgoiti

Similarité par SSD

On va chercher des correspondences de points entre les images obtenues par les caméras de gauche et de droite. Dans l'image de droite, les objects apparaissent un peu plus a la gauche (fig. 1).

FIGURE 1 – Images gauche et droite

Alors, pour retrouver un pixel de l'image de gauche dans l'image de droite, il faut le rechercher dans valeurs égales ou plus petites de l'axe x (et l'inverse pour un pixel de l'image de droite dans l'image de gauche). Pour savoir si deux pixels des images correspondent au même point du monde réel, on examinera leur environnement et le comparera en utilisant la somme de distances carrées. La fonction iviComputeLeftSSDCost calcule la SSD entre deux régions des images en prennant en compte un décalage donné et une taille de la fenetre. Dans un premier temps, on va calculer la SSD pour décalages entre 0 et 32, et enregistrer chaque resultat dans mSSD. La variable mMinSSD contiendra le minimum mSSD pour chaque pixel. Le décalage correspondant a mMinSSD sera enregistré dans mLeftDisparityMap.

Accès aux pixels

Grace à ces instructions

```
pdPtr1 = mMinSSD.ptr<double>(iRow);
pdPtr2 = mSSD.ptr<double>(iRow);
pucDisparity = mLeftDisparityMap.ptr<unsigned char>(iRow);
```

On prends des pointeurs aux débuts des images. En utilisant l'operateur increment, on change leur position selon leur type.

La taille de la démi-fenêtre est utilisée pour obtenir le début de chaque ligne.

```
pdPtr1 += iWindowHalfSize;
pdPtr2 += iWindowHalfSize;
pucDisparity += iWindowHalfSize;
```

Les pixels etant plus proches du bord de l'image que la taille de la démi-fenetre n'ont pas assez pixels autour d'eux (fig. 2), donc il faut commencer par le premier pixel ayant suffisament de pixels dans son environnement.

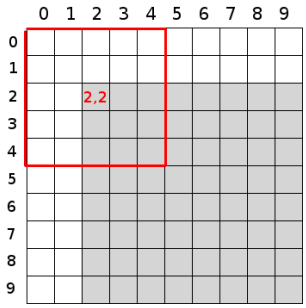

FIGURE 2 – Pixels dont la SSD va être calculée

La fonction mixMaxLoc nous permettre de retrouver les valeurs minimale et maximale des images. La fonction normalize modifie tous les pixels de telle façon que les nouveaux minimum et maximum correspondent à les valeurs indiquées. En utilisant 0 et 255 on maximise le contraste. (figs. 3, 4)

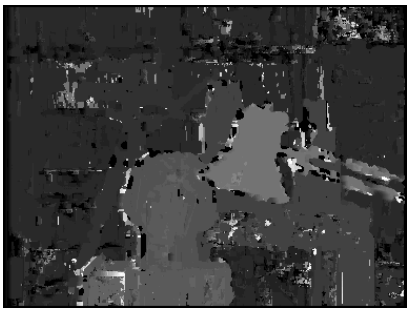

FIGURE 3 – Carte de disparités gauche

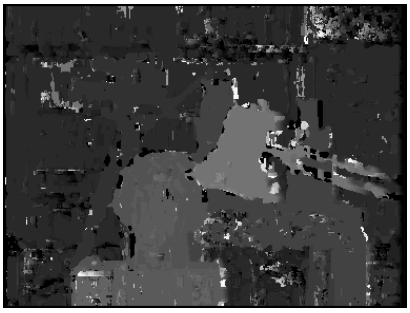

FIGURE 4 – Carte de disparités droite

On voit qu'il y a de décalages apparement bizarres là où il peut y avoir des occultations.

Vérification gauche-droite

Pour minimiser le nombre de faux appariements, on peut conserver seulement les correspondences satisfaisant cette proprieté

$$\hat{d}_r(x_r, y) = \hat{d}_l(x_r + \hat{d}_r(x_r, y), y)$$
$$\hat{d}_l(x_l, y) = \hat{d}_r(x_l - \hat{d}_l(x_l, y), y)$$

c'est à dire, si un pixel a est lié a un pixel b selon la carte de disparités gauche, le pixel b doit être lié au pixel a selon la carte de disparités droite.
Après le verifier, on obtient une masque de validité pour la carte de disparités (fig. 5).

FIGURE 5 – Masque de validité (taille démi-fenêtre=2)

De la carte de disparités, on ne conservera que les pixels dont la valeur de la masque est 0. On se rend compte, quand même, qu'il y a un grand nombre de faux appariements (pas seulement les occultations). En utilisant une plus grande fenêtre de correlation ($w = 4$) on obtient plus d'information sur l'environnement de chaque pixel, alors la probabilité de retrouver un vrai appariement augmente (fig. 6). C'est increment rend l'algorithme beaucoup plus lourd, mais ça ne pose plus un problème avec la mèthode recursif décrit ensuite.

Calcul efficace de SSD

Dans le rapport, les autheurs prennent en compte la redondance presénte dans les calculs. Cette redondance est aussi presénte dans la méthode vu en cours. Pour calculer la SSD d'un pixel, on fait des calculs déjà fait en calculant le pixel précédent. Dans la figure 7 on voit que une grande partie des calculs effectués pour le pixel $(2, 2)$ sont faites encore une fois pour la SSD du pixel $(2, 3)$.

FIGURE 6 – Masque de validité (taille démi-fenêtre=4)

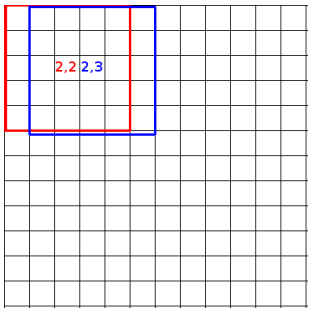

FIGURE 7 – Calculs redundants

Pour eviter ça, les autheurs proposent de ne calculer que les nouvelles différences carrés. C'est evident qu'il ne faut que faire des calculs pour les pixels cochés en vert dans la figure 8

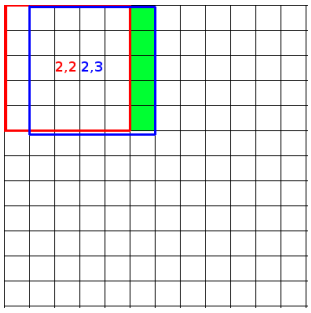

FIGURE 8 – Pixels à calculer

De plus, les autheurs utilisent aussi la recursivité pour eviter des répétitions en calculant les valeurs des colonnes de pixels.
On peut utiliser cette mèthode pour le calcul de la SSD en modifiant la fonction P.

$$P(x, y, d) = (I_l(x, y) - I_r(x - d, y))^2$$

Le nombre de calculs est determiné principalement par les appels recursifs à la fonction $Q(x, y, d)$. Lorsque $y = 0$, la fonction Q calcule la valeur de P pour tous les pixels compris dans

l'hauteur de la fenêtre (une partie reste en dehors l'image, donc c'est ignorée). Pour les autres cas, la fonction calcule une valeur et enleve une autre, c'est à dire qu'elle fait deux soustractions et une multiplication.

La fonction Q fait, alors, $y * 3 + w + 1$ opérations élémentaires (w etant la taille de la démi-fenêtre). Etant donné que pour calculer la SSD d'un pixel il faut appeller la fonction Q deux fois et additioner la valeur précédente, chaque pixel aura besoin de $2 * (y * 3 + w + 1) + 1$ opérations élémentaires.

La mèthode vu en cours a besoin de $3 * (2w + 1)^2 - 1$ opérations pour chaque pixel (Différences entre chaque paire, puissance et addition de toutes les valeurs). L'optimisation recursif dépends linéairement de la taille de l'image, mais la mèthode originale dépends exponentiellement de la taille de la fenêtre de correlation.

C'est evident, alors, que la mèthode optimisée nous permet d'utiliser des fenêtres beaucoup plus grandes sans pénalisation.