EDT in $\mathbb{Z}^3$

8 janvier 2002

1 Notations

On note $F = \{f_{ijk}\}$ l'image binaire $(L \times M \times N)$ dans laquelle les 0-voxels sont le fond et les 1-voxels l'objet. On cherche la carte $D = \{d_{ijk}\}$ sur les voxels de l'objet donnant la distance Euclidienne au plus proche voxel du fond. On note $S = \{s_{ijk}\}$ la distance Euclidienne au carré.

On a donc :

$$s_{ijk} = min_{(p,q,r)}\{d((i,j,k),(p,q,r))^2\} \quad (1)$$

$$f_{pqr} = 0 | \{1 \leq p \leq L, 1 \leq q \leq M, 1 \leq r \leq N\} \quad (2)$$

$$= min_{(p,q,r)}\{(i-p)^2 + (j-q)^2 + (k-r)^2\} \quad (3)$$

$$d_{ijk} = \sqrt{s_{ijk}} \quad (4)$$

L'algorithme procede dimension par dimension, par exemple on va minimiser d'abord les x, puis reinjecter le calcul dans la minimisation des y puis on minimisera les z.

2 Algo 3D de Saito

Phase 1 : on calcule la transformée en distance en ligne dans l'image, on obtient donc la carte g_{ijk} de distance :

$$g_{ijk} = min_x\{(i-x)^2; f_{xjk} = 0, 1 \leq x \leq L\}$$

Phase 2 : On considère maintenant les colonnes de l'image et on calcule la carte h_{ijk} :

$$h_{ijk} = min_y\{g_{iyk} + (j-y)^2; 1 \leq y \leq M\}$$

Phase 2 : On considère maintenant les hauteurs de l'image et on obtient la carte s_{ijk} :

$$s_{ijk} = min_z\{h_{ijz} + (k-z)^2; 1 \leq z \leq N\}$$

L'algorithme donne bien la transformée en distance car en gros :

$$min_z\{min_y\{min_x\{(i-x)^2\}+(j-y)^2\}+(k-z)^2\} = min_{xyz}\{(i-p)^2+(j-q)^2+(k-r)^2\}$$

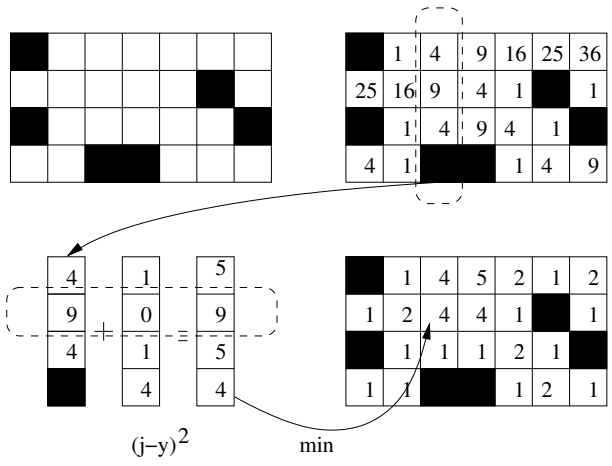

Figure 1: EDT de Saito en 2D

complexité La phase 1 coûte $O(L)$ mais les autres minimisations sont en $O(M^2)$ et $O(N^2)$. On peut cependant réduire ce calcul et obtenir une complexité globale en $O(LMNAv(d_{ijk}))$ où $Av(d_{ijk})$ correspond à la distance moyenne dans l'image.

3 Algorithme modifié

On essaye de réecrire les phases de fusion de dimensions dans une approche basée sur le diagramme de Voronoï. On ne mémorise plus la distance au carrée mais les coordonnées du voxel de fond le plus proche.

Phase 1 : On étiquette chacun des pixels par le voxel du fond le plus proche en ligne. On obtient donc une carte :

$$g_{ijk} = \{(x, j, k) | min_x\{(i - x)^2; f_{xjk} = 0, 1 \leq x \leq L\}$$

Phase 2 : On considère une colonne dans l'image. Chaque voxel de cette ligne pointe vers le voxel du fond le plus proche dans sa ligne. Nous avons donc L sites possibles pour la colonne donnée. On se trouve donc dans le cas de l'algorithme de Breu *et al.* pour la transformée en distance 2D. En effectue donc :

- Calcul des sites pour la colonne en partant des sites en ligne avec le prédicat `remove`
- Étiquetage des voxels par les sites obtenus.

Nous obtenons donc le diagramme de Voronoï 2D sur chacune des coupes.

Phase 3 : On réutilise le même principe pour les hauteurs et dans les dimensions supérieures.

Sur une hauteur h, nous avons N sites qui correspondent aux diagrammes de Voronoï dans chacune des coupes. A partir de ces sites, on reconstruit le diagramme 3D. On a une site par coupe, on peut donc utiliser un `remove_nD` que l'on peut toujours définir. On calcule donc la liste des sites qui vont intervenir pour la hauteur h et on réécrit par la suite.

Complexité Quelques soit la dimension, le prédicat `remove_nD` qui permet de décider si un site est caché par deux autres sites lors de l'intersection avec la droite peut se calculer en temps constant. La procédure qui extrait les sites pour la droite est linéaire au nombre de candidats. La réécriture des la droite étant donné les sites se fait en 2 scans dans toutes les dimensions. La complexité globale est donc linéaire au nombre de voxels $O(LMN)$.

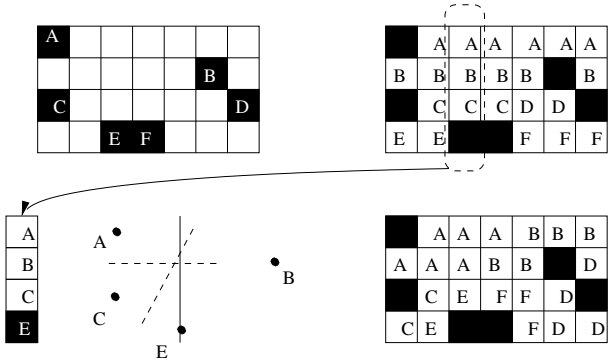

Figure 2: EDT modifié 2D, pour la colonne analysée, les sites C et B seront cachés par A et E.

3.1 Exemple de `remove_3D`

Dans notre cas on a la droite:

$$\begin{cases} x = r \\ y = t \end{cases}$$

On a :

$$\|u - \hat{uv}\|^2 = (u.x - r)^2 + (u.y - t)^2 + (u.z - \hat{uv}.z)^2 \quad (5)$$

$$\|v - \hat{uv}\|^2 = (v.x - r)^2 + (v.y - t)^2 + (v.z - \hat{uv}.z)^2 \quad (6)$$

Supposons $v.z \neq u.z$, alors $\|u - \hat{uv}\| = \|v - \hat{uv}\|$ s'ecrit :

$$\hat{uv}.z = \frac{(v.x - r)^2 + (v.y - t)^2 - (u.x - r)^2 - (u.y - t)^2}{2(v.z - u.z)}$$

Ensuite, $\hat{uv}.z \geq \hat{vw}.z$ si et seulement si:

$$(w.z - v.z)((v.x - r)^2 + (v.y - t)^2 - (u.x - r)^2 - (u.y - t)^2) \quad (7)$$
$$\geq (v.z - u.z)((w.x - r)^2 + (w.y - t)^2 - (v.x - r)^2 - (v.y - t)^2) \quad (8)$$

Ce test constitue le corps du prédicat `remove_3D(u,v,w,r,t)`. Ce prédicat permet de tester si le site v sera caché par les sites u et w sur la droite $x = r, z = t$ sous l'hypothèse que les sites n'appartiennent pas à la même coupe et dans l'ordre $u.z < v.z < w.z$.

3.2 Calcul des sites à partir des candidats

A une dimension donnée, on a un vecteur de sites possibles (*candidats*) et une procédure `remove` adaptée à la dimension courante.

L'algorithme qui calcule les sites pour un segment donné est le suivant:

Algorithme : Calcul des sites à partir des candidats

```
L[1]:= candidat[1]
L[2]:= candidat[2]
k:=2 ; l:=3 ; c:=|candidat|
Tant que l<=c faire
   w:=candidat[l]
   Tant que k >=2 et remove(L[k-1],L[k],w,r,t) faire
      k:=k-1
   finTantque
   k:=k+1
   l:=l+1
   L[k]=w
finTantque
```

Cet algorithme est en $O(|candidat|)$.

3.3 Réecriture

Une fois que la liste des sites intervenant pour le vecteur courant est créée, on étiquette les voxels avec la bon site en parcourant les sites obtenus. Ce calcul est linéaire au nombre de cases dans le vecteur (les sites vont apparaître dans le vecteur dans le même ordre que la liste L précédente).

3.4 Complexité globale

Pour chaque dimension :

Création de la liste des sites	$O(n)$
Réécriture du vecteur	$O(n)$
Étiquetage en distance	$O(n)$

Donc calcul linéaire au nombre de voxels. En mémoire on ne travaille que sur une image de même taille avec des pointeurs vers les pixels du fond.