R5.06 - Traitement de l'image

TP4 : Rotation d'images

Contenu de la Page

Principe

retour haut de page

Nous voulons faire une rotation plane d'une image autour de son centre *O* avec un angle *a* en degré. Si *(x,y)* sont les coordonnées cartésiennes d'un point *M* alors les coordonnées cartésiennes *(x',y')* du point M', résultant d'une telle rotation, se calculent selon la formule :

x' = x.cos(a)+y.sin(a) et *y' = -x.sin(a)+y.cos(a).*

Les problèmes qu'on rencontre

retour haut de page

En Java, le cosinus et le sinus se calculent avec des angles exprimés en radians, mais `Math.toRadians(.)` permet de convertir les degrés en radians.

Le deuxième problème rencontré est de calculer correctement la taille de l'image résultante, une feuille de papier permet de comprendre que le calcul n'est pas si compliqué, vu la formule donnée.

Le troisième problème est que les coordonnées de l'image commence en haut et à gauche, et non au centre de l'image. Si *(i,j)* sont les coordonnées de l'image avec l'origine en haut et à gauche alors $(x,y) = (i-i_0 ,j-j_0)$ sont les coordonnées du même point mais avec l'origine au centre, (i_0 ,j_0) sont les coordonnées du centre de l'image.

Le quatrième problème est que même si *(x,y)* sont des coordonnées en entier, il y a peu de chance que *(x',y')* soient des coordonnées en entier. Le 3ème problème est que les coordonnées *(x',y')* sont calculés avec l'origine au centre, or les coordonnées dans la nouvelle image sont avec l'origine en haut et à gauche. On a le deuxième problème à l'envers !

Méthodes Java utiles

retour haut de page

- `getWidth()`,
- `getHeight()`,
- `toRadians(.)`,
- `sin(.)`,
- `cos(.)`,
- `round(.)`,
- `abs(.)`,
- `setRGB(.)`,
- `getRGB(.)`.

Exercice 1 : Rotation naïve

retour haut de page

On choisit le pixel de la source et on calcule le pixel de la destination correspondant le plus proche. L'implémentation parcourt la source.

Travail à Faire

1.1 Créez une telle méthode, testez avec différents angles. Que remarque-t-on ?
1.2 Quel est le problème ? Pourquoi ?

Exercice 2 : Rotation par échantillonnage (RSamp)

retour haut de page

On choisit le pixel de la destination pour avoir la valeur du pixel source correspondant le plus proche. L'implémentation parcourt la destination.
Donc ici, à partir de *(x',y')*, on veut *(x,y)*, il suffit d'utiliser les formules suivantes : *x = x'.cos(a)-y'.sin(a)* et *y = x'.sin(a)+y'.cos(a).*
On remarque que le code est très similaire à la première méthode, mais on peut avoir des coordonnées qui ne sont pas dans l'image d'origine.

Travail à Faire

2.1 Créez une telle méthode, testez avec différents angles. A-t-on le même résultat que la première méthode ?
2.2 Est-ce qu'on n'aurait pas pu optimiser ceci pour les angles : 90°, 180° et 270° ?
2.3 Que se passe-t-il si on applique plusieurs rotations successives sur une image ?
Donnez 2 solutions différentes pour résoudre ce problème.

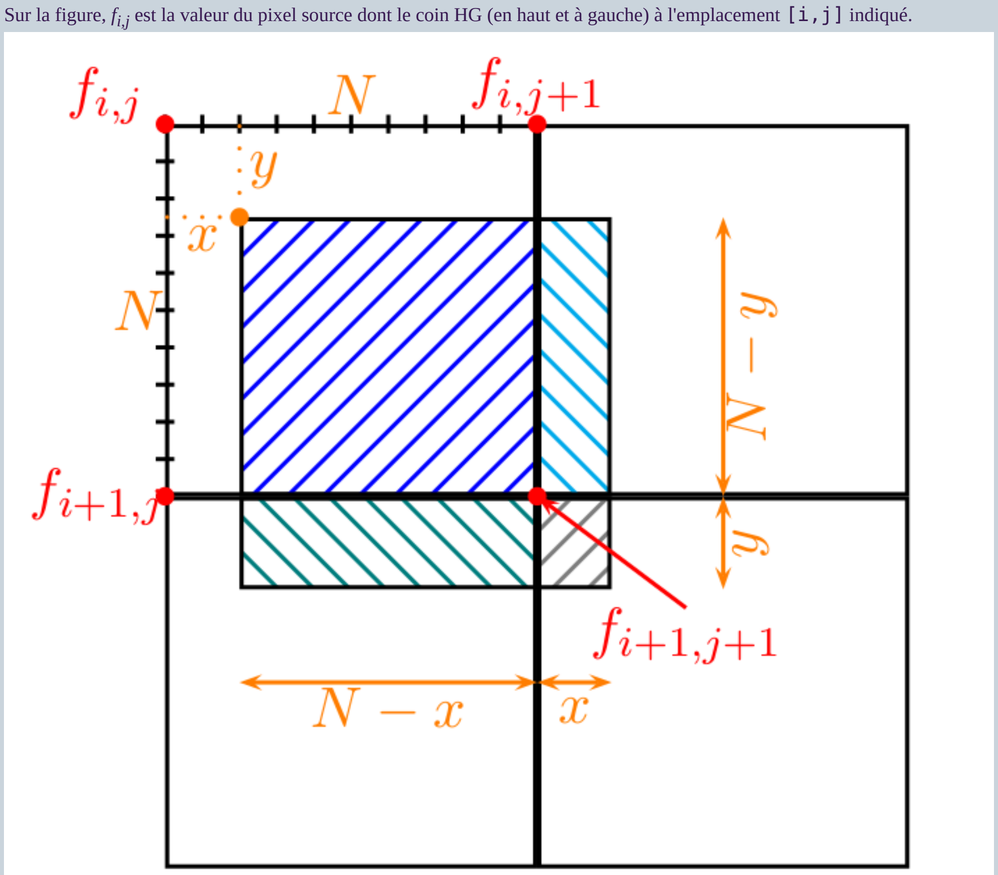
Exercice 3 : Rotation par cartographie de zone (Rotation by Area Mapping)

retour haut de page

La RAM est limitée dans son application aux situations où vous avez une image en niveaux de gris de 8 bpp ou une image RGB de 24 bpp, vous voulez la meilleure qualité d'image pivotée et vous pouvez sacrifier une certaine vitesse pour la qualité.

La cartographie de zone fonctionne comme suit. Pour chaque pixel cible, vous trouvez les 4 pixels source qu'il recouvre partiellement. Vous calculez ensuite la valeur du pixel destination comme la moyenne pondérée en surface de ces 4 pixels source. On fait deux approximations simplificatrices :
- Les zones sont calculées comme si le pixel de destination était translaté mais pas tourné.
- La zone de chevauchement est calculée sur une grille de sous-pixels discrète. Par exemple pour une couleur décomposée en composantes de 8 bpp avec 256 niveaux, il est pratique de diviser chaque pixel en une grille de sous-pixels de 16x16 et de compter le nombre de sous-pixels superposés.

Sur la figure, $f_{i,j}$ est la valeur du pixel source dont le coin HG (en haut et à gauche) à l'emplacement `[i,j]` indiqué.



Quatre pixels source sont affichés, chacun avec sa valeur de pixel étiquetée dans son coin HG. Chaque pixel source est subdivisé en NxN sous-pixels; nous montrons la subdivision seulement sur un des pixels. Nous voulons déterminer la valeur du pixel destination dont le coin HG est à l'emplacement V, qui a des coordonnées *x* et *y*, en unités de sous-pixels, par rapport au coin HG du pixel source supérieur gauche. En prenant les contributions des quatre pixels source dans l'ordre HG, HD, BG, BD, la moyenne normalisée du pixel de destination, en utilisant la moyenne de zone, est :
Valeur du pixel cible = $(1/N^2)\ [(N - x).(N - y).f_{i,j} + x.(N - y).f_{i,j+1} + (N - x).y.f_{i+1,j} + x.y.f_{i+1,j+1}\ ]$

Il s'agit du même filtre numérique que celui obtenu pour l'interpolation linéaire lors de la mise à l'échelle arbitraire d'images en niveaux de gris !

On doit faire ce calcul pour chacune des composantes R, G et B de la couleur du pixel, et recomposer la couleur moyenne.

Travail à Faire

3.1 Créez une telle méthode, testez avec différents angles.
3.2 Comparez des détails de l'image entre Rsamp et RAM, que peut-t-on dire ?