UNIVERSITÉ SAVOIE MONT BLANC

Projet C++ : Image : Segments

Introduction à c++

—MMAA—

1 Position du problème

Une image peut être vue comme une succession de pixels disposés les uns à côté des autres. La vision par ordinateur, la synthèse d'images et le traitement d'images sont trois domaines qui tentent d'analyser ces images et d'en tirer des informations. L'objectif peut être de détecter le contour des objets, ensuite de détecter des objets simples comme des droites ou des cercles puis des objets plus complexes commme des coniques et enfin des objets quelconques comme un visage, une voiture...
Dans le cadre de ce projet, nous nous limiterons aux segments de droites. Ces segments peuvent être de n'importe quelle taille et de n'importe quelle épaisseur, pourvu qu'ils soient composées d'un ensemble de points à peu près alignés.

2 Méthode

La détection de segments va s'effectuer en trois parties :

- Appliquer la transformée de Hough dans l'espace des paramètres (m, p) sur une image avec un format donné (voir Section 3).
- Appliquer la transformée de Hough dans l'espace des paramètres (r, θ) sur une image avec un format donné (voir Section 3).
- Les paramètres des droites à détecter ne sont malheureusement pas toujours très précis et plusieurs résultats représentants la même droite sont souvents trouvés. La troisième partie consiste à supprimer les doublons et à choisir les meilleures droites.

3 Transformation de Hough

3.1 Principe en bref

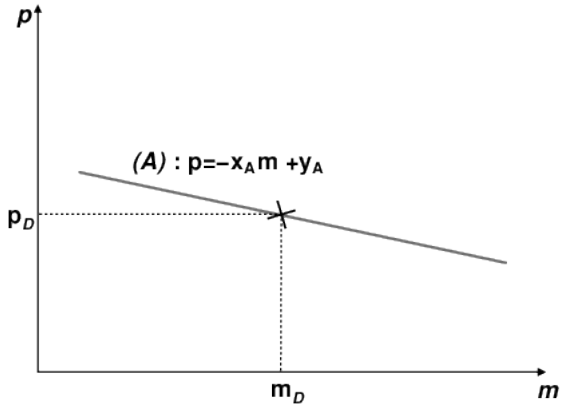
Soit (D) une droite notée $y = mx + p$ passant par deux points distincts A et B. La droite (D) vérifie donc les équations :

$$y_A = mx_A + p \tag{1}$$
$$y_B = mx_B + p \tag{2}$$

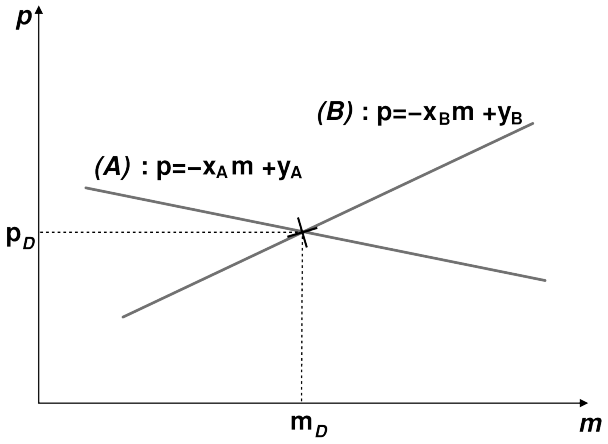
La transformation de Hough propose de changer d'espace de représentation de ces droites et de ces points de la manière suivante :

$$(A) : p = -x_A m + y_A \tag{3}$$
$$(B) : p = -x_B m + y_B \tag{4}$$

L'équation (1) engendre la droite (A) (équation(3)) de coefficient directeur $-x_A$ et d'ordonnée à l'origine y_A. Cette droite est associée spécifiquement au point A. La transformation de Hough d'un point donne donc une droite. La droite (A) passe par une infinité de couples (m_i, p_i) de points. Ces couples de points de l'espace (m, p) correspondent à toutes les droites passant par le point A dans l'espace (x, y).



En faisant de même avec le point B, on obtient la droite (B) qui passe elle aussi par le point (m_D,p_D). Ainsi, n points alignés dans l'espace (x, y) génèrent n droites sécantes dans l'espace (m, p).



Concrètement, la transformée de Hough est stockée dans "un buffer d'accumulation" c'est-à-dire un tableau à deux dimensions dans le cas de l'espace (m, p) sur lequel on dessine les droites associées aux points de l'espace (x, y). A chaque case de ce tableau on attribue un score variant selon le nombre de fois qu'elle a été parcourue par une droite. Ceci implique qu'un point intersecté par deux droites a un score supérieur à un point parcouru par une seule droite.
Pour trouver les droites présentes dans l'espace (x, y), il suffit de chercher les pixels de score maximal dans l'espace (m, p). Les pixels (m_i, p_i) choisis nous indiquent qu'un grand nombre de points sont alignés sur la droite $y = m_i x + p_i$ dans l'espace (x, y).

3.2 Autre espace de paramètres

En utilisant la méthode décrite précédement, on s'aperçoit rapidement que pour la plupart des points de l'espace (x, y), les coefficients directeurs des droites obtenues sont très élevés et que pour certains points, l'ordonnée à l'origine est telle que la droite n'apparait pas forcément dans le tableau (de taille finie). En représentant une ligne le la façon suivante, ce genre de problèmes disparait :

$$r = x.cos\theta + y.sin\theta \tag{5}$$

$$\text{avec } \theta \in [0, 2\pi[$$

L'ensemble des couples (r_i, θ_i) vérifiant l'équation $r = x_A.cos\theta + y_A.sin\theta$ correspond à l'ensemble des droites passant par A en représentation polaire. De même, tous les couples (x_i, y_i) vérifiant l'équation $r_D = x.cos\theta_D + y.sin\theta_D$ appartiennent à la droite (D).

Deux remarques :

- Il existe une valeur R telle que si $|r| > R$, la droite en question ne figure pas sur l'image originale. Autrement dit une des dimensions du "buffer d'accumulation" est fixée.
- L'angle θ varie de 0 à 2π, toutefois sur cet intervalle, la même droite est représentée deux fois : une première fois avec (r, θ) et une seconde fois avec $(-r, \theta + \pi)$. Il est donc

suffisant de traiter θ sur l'intervalle $[0, \pi[$. La seconde dimension du "buffer" est donc elle aussi fixée.

Ce qui vient d'être fixé est l'échelle du "buffer d'accumulation" et non sa taille en nombre de cases. Celle-ci doit être choisie judicieusement car si le nombre de cases est trop faible, les équations de droites trouvées seront imprécises et si le nombre de cases est trop élevé, la détection deviendra longue et un nombre important de droites seront proposées pour représenter la même droite dans l'image originale.

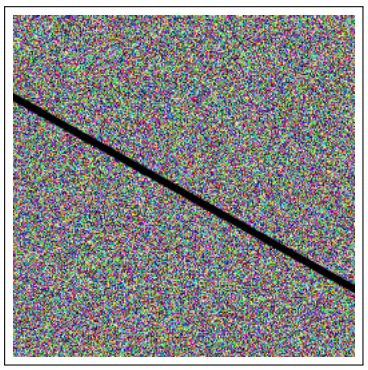
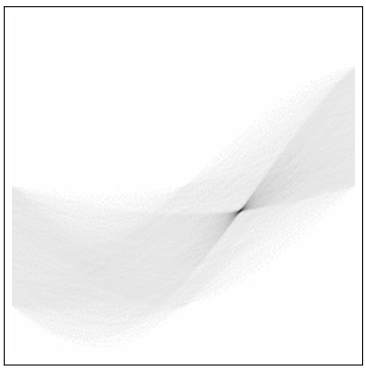
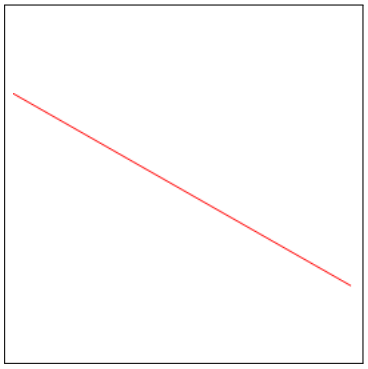


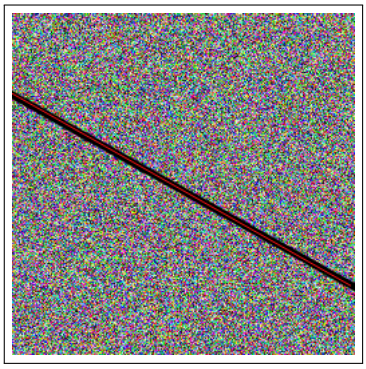
Image originale



Transformation de Hough



Droite détectée



Droite sur l'image originale

4 Suppression des doublons et choix de la meilleur droite

La transformation de Hough permet de détecter la présence de droites dans une image mais parfois propose plusieurs droites pour représenter en fait la même droite. Prenons par exemple le cas d'une ligne épaisse de quelques pixels, sa transformation de Hough proposera plusieurs lignes de un pixel de large les unes à coté des autres, or seulement une droite est représentative de cette ligne.
Ce problème vient du fait que les pixels représentant une droite sont rarement alignés et que si la droite en question fait plusieurs pixels de large, elle correspond effectivement à plusieurs droites. Ceci se traduit par un point d'intersection des courbes étalé sur plusieurs cases du "buffer d'accumulation". La case de score maximal n'est pas forcément la plus représentatrive de la droite et un calcul de barycentre sub-pixel peut par exemple donner un résulat plus précis.

5 Format de l'image et le tracé de droites

5.1 Format requis

Vous allez devoir manipuler les fichiers *.ppm* reconnu par divers éditeurs d'images comme *photoshop* ou *gimp*. Le format le mieux adapté à notre problème est le format couleur en binaire (P3). Le fichier commencera donc avec un en-tête correspondant.

Comme vu pendant le projet de Python, un pixel en couleur est défini par 3 nombres : un pour "la quantité" de rouge (R), un pour "la quantité" de vert (V) et un pour "la quantité" de bleu (B), noté RVB (RGB en anglais). Nous coderons ces quantités par des `uint8_t`. Un `uint8_t` est codé sur 8 bits, soit un octet. Ces nombres vont donc de 0 à 255.

Revenons au format *ppm*. Le premier nombre correspond au "numéro de variante" choisi, dans notre cas $P6$. Sur la ligne suivante, les deux nombres correspondent à la *largeur* et la *hauteur* de l'image. Enfin, le dernier nombre correspont à la valeur maximale que l'on puisse rencontrer (ici 255). Des commentaires peuvent être insérés dans l'en-tête, une ligne de commentaires commence par un #.
Chaque pixel sera écrit dans le fichier en procédant ligne par ligne, de gauche à doite, en commençant par le haut. Les pixels seront écrits sous forme de 3 valeurs écrits les uns sous les autres (RVB).

exemple:

```
#commentaires
P3
# dimensions de l'image
30 30
# valeur max
255
# valeurs de RVB
255
255
255
128
128
128
...
```

Un exemple complet d'image est donné sur la page moodle du cours.

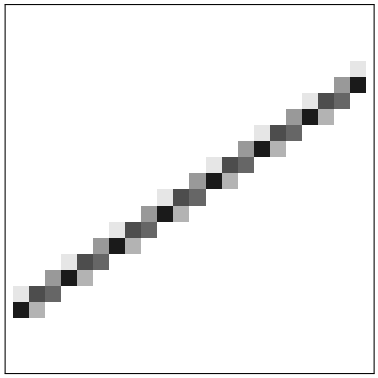
5.2 Dessiner une droite

Ce projet ne nécessite **aucune bibliothèque graphique**, il faut cependant pourvoir tracer une droite sur une image *.ppm*. Le paragraphe suivant explique la méthode utilisée pour

dessiner une droite. Soit $(D) : y = m.x + p$ la droite à tracer, la première étape, appelée *line clipping*, sert à délimiter parmi toute la droite la partie visible sur l'image. On ne trace en effet qu'un segment $[A, B]$ de cette droite dont il faut déterminer les coordonnées. Une fois les points A et B déterminés, il faut tracer le segment $[A, B]$. Soit $q_{i,j}$ le pixel situé sur la colonne j de la ligne i, une première approche serait de parcourir les colonnes de $j = x_A$ à x_B et de dessiner le pixel $q_{m.j+p,j}$. Malheureusement certains segments sont très mals dessinés avec cette méthode, notament les segments presque verticaux qui ne sont alors représentés que par deux ou trois pixels. Il y a donc deux cas à différencier :

- Soit $|m| \leq 1$ dans quel cas la méthode précédente est satisfaisante.
- Soit $|m| > 1$ et il est alors préférable de parcourir les ligne de $i = y_A$ à y_B et de dessiner les pixels $q_{i,\frac{i-p}{m}}$.

Il est très fortement recommandé de gérer l'anti-aliasing c'est à dire que si le point à colorier ne tombe pas exactement au milieu d'un pixel, il faut "partager" la couleur avec les pixels voisins.



6 Travail demandé

Le but du projet est d'implémenter en langage c++ un programme permettant de détecter des segments (au format ppm). Votre programme sera composé de fichiers `.cpp` et `.hpp`. Il se compilera avec `g++`.

Les trois parties listées dans la Section 2 devront être programmées. A l'issue du projet, l'étudiant ou le groupe d'étudiants doit remettre un dépôt (github) présentant l'avancement de son travail, le code développé, un `Readme.md` expliquant le sujet, décrivant les points saillants du travail réalisé ainsi que comment se servir du code développé. Des exemples d'utilisation doivent aussi être donnés.