TREIZIÈME COLLOQUE GRETSI - JUAN-LES-PINS DU 16 AU 20 SEPTEMBRE 1991

ALGORITHME RAPIDE POUR L'EXTRACTION DE SEGMENTS DE DROITE DANS LES IMAGES DE CONTOUR

Y. RUNGSUNSERI et A. CHEHIKIAN

Laboratoire de Traitement d'Images et Reconnaissance de Formes
I.N.P.G. 46 Av. Félix Viallet 38031 Grenoble Cédex

RÉSUMÉ

Nous proposons un algorithme rapide pour réaliser l'extraction de segments de droite en un seul passage dans une image. L'algorithme intègre la phase intermédiaire de chaînage et procède à l'approximation polygonale directement sur cette image. Il exploite pour cela, l'information de contexte à partir du 8 voisinage d'une part, l'information d'orientation des contours d'autre part. Il opère la fusion de chaînes scindées par le sens de balayage et élimine les petites chaînes considérées comme bruit, sans avoir recours à des étapes supplémentaires. Nous illustrons l'efficacité de cet algorithme par quelques exemples sur des images réelles.

ABSTRACT

We propose a fast algorithm for straight line extraction in images with a single pass raster scan. The algorithm integrates the intermediat phase of linking process and directly performs polygonal approximation in the image. For that, it exploits the context information on the eight neighborhoods, and the orientation information of edge pixels. The result is an optimal set of straight lines whose length exceeds a given threshold, without any post-processings as merging collinear lines and eliminating noisy lines. We illustrate the efficiency of this algorithm by means of some examples on real images.

1. Introduction

Une étape fondamentale dans la plupart des systèmes de vision par ordinateur est d'engendrer une description compacte d'une image, plus exploitable que l'ensemble de pixels. Cette description peut être sous forme de régions, courbes, segments de droite, etc. Nous nous intéressons dans cet article à l'extraction de segments de droite dans les images de contour.

Les algorithmes proposés par différents auteurs [1, 2, 3] se basent sur le même principe. Le processus se compose de deux phases :

1. *Chaînage de contour* qui consiste à relier tous les pixels connexes entre eux sous forme de chaînes. La coupure d'une chaîne se trouve aux pixels de jonction, où a lieu l'intersection des chaînes.
2. *Approximation polygonale* qui consiste à approcher chaque chaîne par des segments de droite, en respectant un critère choisi.

Cette approche est performante au point de vue des résultats obtenus, mais très coûteuse en temps de calcul. Pour notre part, nous désirons plutôt un algorithme rapide qui soit réalisable en temps réel pour un coût réduit, et avec une performance comparable.

Pour réaliser cela, nous avons choisi de faire fonctionner notre algorithme en un seul balayage sur l'image, autrement dit un algorithme particulier de chaînage qui procède à l'approximation polygonale.

2. Analyse du problème

L'algorithme de chaînage proposé par Giraudon [4] est représentatif pour le chaînage de type un seul balayage sur l'image de contour. Nous analysons quelques problèmes résidant dans cet algorithme, au point de vue d'une extraction de segments de droite, ci-dessous :

1. La création des chaînes en un seul passage a pour conséquence de scinder certaines chaînes en plusieurs morceaux alors que les pixels chaînés sont alignés sur une même droite. Des exemples caractéristiques sont donnés à la figure 1.

a. *b.*

Fig.1 *Segments de droite scindés en plusieurs morceaux.*

2. Beaucoup de chaînes stockées sont trop petites pour être prises en compte dans les traitements ultérieurs, car une chaîne est créée dès qu'un premier pixel est rencontré.

Ces problèmes ont été résolus en ajoutant des étapes supplémentaires (fusionner et éliminer) sur les chaînes obtenues.

Par ailleurs, cet algorithme n'exploite que l'information sur la configuration des pixels de contour. Sur une fenêtre utilisée de taille 3x3, l'algorithme est capable de détecter les jonctions potentielles où se connectent au moins 3 chaînes.

Mais, avec telle information, on ne pourra pas détecter des points de cassure entre 2 segments de droite, dans des coins carrés par exemple.

3. Idée de base

Notre algorithme pour l'extraction de segments de droite est basé sur deux hypothèses :

1. La description d'un segment de droite, représentatif d'un ensemble des pixels chaînés, par les deux points d'extrémité est acceptable dans la plupart des applications.
2. L'information d'orientation des pixels de contour est fiable et peut être retenue comme un critère d'appartenance à un même segment.

Ces hypothèses sont particulièrement vérifiées dans des scènes industrielles. Nous utilisons l'algorithme de Canny [5] pour générer l'image de contour. L'orientation de chaque pixel de contour est discrétisée en n valeurs et codée par 0, 1, ..., n-1, comme le montre la figure 2 où $n = 8$.

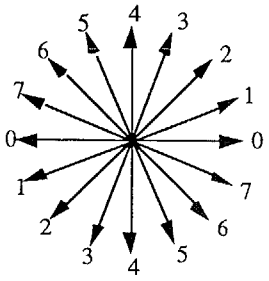

Fig.2 *Discrétisation en 8 codes*

L'idée de base de l'algorithme est d'une part de regrouper dans une même chaîne tous les pixels de contour possédant le même code d'orientation avec une tolérance d'un saut de code, afin de ne pas scinder les segments orientés vers la frontière entre deux codes. Donc, chaque chaîne résultant de l'algorithme peut être constituée de codes bornés dans (0,1), (1,2), ..., (n-1,0). D'autre part, à chaque pixel inclut dans une chaîne, il n'est plus nécessaire de mémoriser tous les pixels participant. Seuls les points d'extrémité sont significatifs. Cela facilite énormément la gestion des données.

De cette façon, le nombre n dans la discrétisation de l'orientation joue un rôle important dans le contrôle des erreurs d'approximation polygonale. Expérimentalement, n égale à 8 ou 16 est recommandé.

P1	P2	P3
P4	Pr	F1
F2	F3	F4

P1-P4 : passé
Pr : présent
F1-F4 : futur

Fig.3 *Classification des pixels dans une fenêtre 3x3.*

4. Gestion des chaînes

En balayant l'image de manière séquentielle de haut en bas et de gauche à droite, on distingue les pixels dans une fenêtre 3x3 en 3 classes : passé, présent et futur (Fig.3). L'information sur les deux derniers est celle portée dans l'image. Par contre, l'information sur le passé est sous forme des chaînes qui sont connexes du pixel central. On définit pour cela une structure proche de la structure image : un tableau qui recouvre 2 lignes de l'image et de la même dimension en colonne que l'image, appelé PIT (Predecessor Information Table). Chaque pixel du passé (un élément de PIT) est composé par :

- le type de pixel
- les coordonnées des points d'extrémité
- le nombre de pixels participant à la chaîne
- le code chaîne, représente l'orientation moyenne de la Chaîne.

Le type de pixel a pour but de caractériser la chaîne que nous représentons de la manière suivante :

- Chaîne ayant un seul élément
  `[T0]` si on permet une seule continuation sur le pixel.
  `[T1]` si il y aura 2 continuations.
- Chaîne horizontale ayant plus d'un élément et pouvant être continuée en deux extrémités
  `[T2]` pour l'extrémité à gauche, et toujours avec
  `[Q1]` pour l'autre extrémité (à droite).
- Chaîne ayant plus d'un élément et pouvant être continuée en une seule extrémité (même les chaînes horizontales)
  `[Q0]` signifie la détection définitive de l'autre extrémité.
  `[UU]` pour neutraliser les pixel non-significatifs.

5. Principe de l'algorithme

Il s'agit de créer et gérer les chaînes en parallèle avec le processus de balayage. Pour chaque pixel de contour rencontré, la connexité du pixel est examinée à partir du 8 voisinage : les pixels du passé dans PIT et les pixels du futur dans l'image. Cette information nous permet de distinguer grossièrement 3 classes de pixel :

Terminal simple	Si le pixel est connexe à un seul voisin.
Jonction potentielle	Si le pixel est connexe à au moins un pixel du passé et à plus d'un pixel du futur, ou vice-versa.
Pixel flou	Ce sont les configurations où on n'a pas assez d'information pour décider si le pixel est un point de cassure. Dans ce cas, on fait un test de continuité basé sur l'information d'orientation. L'idée est déjà présentée dans *§3*.

Selon la configuration, un traitement approprié sera appliqué. Chaque traitement peut être composé par une ou plusieurs opérations de base comprenant *ouvrir*, *continuer*, *fusionner* et *fermer*.

Chaque fois une opération de base est effectuée, les éléments concernant dans PIT sont mis à jour de manière à conserver la représentation de chaînes comme l'on a définie dans *§4*. Une nouvelle chaîne est créée dans PIT par *ouvrir* dès qu'un premier pixel est rencontré. Un pixel est inclus dans une chaîne

existant par *continuer* si il satisfait le test de continuité, ou on ferme la chaîne sinon. L'opération *fusionner* est conçue pour le cas où deux chaînes possédant la même orientation se connectent sur un même pixel. Enfin, l'opération *fermer* a pour neutraliser un pixel du PIT quand un point d'extrémité est détecté. Dès qu'une chaîne est fermée sur toutes les deux côtés, on retire un segment si sa longueur (en nombre de pixels participant) dépasse un seuil τ_L.

Le résultat de l'algorithme, après un seul passage dans une image, est donc une liste de segments de droite. Chacun de ces segments est décrit par ses deux extrémités, avec la possibilité de stockage étendu pour l'information sur l'orientation et la longueur de segment.

6. Réalisation algorithmique

Nous décrivons ci-dessous 3 points particuliers dans la réalisation de notre algorithme :

1) Calcul du code chaîne

Le code chaîne a été introduit dans chaque élément du PIT comme code représentant l'orientation moyenne de la chaîne. Le fait que chaque chaîne ne contient pas plus de deux codes d'orientation qui sont adjacents, nous avons ajouté pour le code chaîne, dans le cas de l'image avec une résolution de n codes, n codes supplémentaires. Un code chaîne prend alors une valeur dans $\{0, 1, \ldots, 2n\text{-}1\}$. Nous représentons une mélange entre (0,1) par n, (1,2) par n+1, ... et (n-1,0) par 2n-1. De cette façon, le calcul d'un code chaîne pour deux codes donnés peut se faire par un tableau de taille $2n$x$2n$.

2) Test de continuité

C'est un test logique sur l'orientation des contours pour décider si l'on peut combiner une chaîne avec un pixel dans le cas de *continuer*, ou avec une autre chaîne dans le cas de *fusionner*. Dans le cas où les deux codes sont différents, on cherche dans un tableau de taille $2n$x4 si le deuxième code est trouvé parmi les 4 nombres dans la ligne correspondant au premier code. En effet, il existe au maximum 4 codes adjacents pour chaque code chaîne, par exemple pour 1 il y a (0, 2, n, n+1), etc.

3) Accès au traitement

A chaque pixel traité, le traitement qui doit être effectué est défini à partir de la connexité de son voisinage. Le voisinage, constitué de 8 points de valeurs binaires, peut prendre 256 valeurs différentes. Nous avons donc utilisé un tableau de 256 éléments adressé par cette valeur. Chacun des éléments du tableau contient un numéro du traitement avec éventuellement un ou deux paramètres codant la position du voisinage concerné. De cette façon l'accès aux traitements est instantané pour un voisinage donné.

En résumé, la réalisation de notre algorithme est faite autour de 3 tableaux. Les deux premiers servent à l'exploitation de l'information d'orientation. Le dernier est modifiable cas par cas, donc permet de rectifier facilement l'algorithme.

7. Résultats expérimentaux

Cet algorithme a été testé sur divers types d'images. Les résultats sont très satisfaisants. Nous illustrons l'efficacité de l'algorithme par deux exemples sur des scènes différentes :

La figure 4 montre les résultats sur une image de femme où résident des contours lisses. Avec τ_L=3, on obtient 538 segments dans Fig. 4b, et 568 segments dans Fig. 4c, en utilisant l'orientation codées en 8 et 16 codes respectivement. Dans cet exemple, on remarque le rôle important de la définition de discrétisation.

Un autre exemple, la figure 5 présente une image du bureau INRIA qui est représentative pour l'application robotique. De même, le nombre de segments est 516 et 543 respectivement dans Fig. 5b et Fig. 5c. On remarque cette fois-ci la moindre différence entre les deux figures, au contraire du cas précédent.

8. Conclusion

Nous avons présenté un algorithme rapide qui accomplit une extraction des segments de droite en un seul balayage de l'image par une fenêtre 3x3.

L'algorithme est défini cas par cas en fonction de la connexité du 8 voisinage. Il utilise par ailleurs l'information d'orientation pour détecter les points de cassure dans le cas du pixel flou.

L'algorithme est simple mais efficace. Les résultats expérimentaux sur divers types d'images ont prouvé sa validité.

Une implantation temps réel a été réalisée sur un microprocesseur de traitement du signal [6]. Il est capable de traiter une image de taille 256x256 en environ 50 milli-secondes.

Bibliographie

[1] Nevatia R. and Babu K. R., "Linear Feature extraction and description", *Computer Graphics and Image Processing*, vol. 13, 1980, pp. 257-269.

[2] Zhou Y. T., Venkateswar V. and Chellappa R., "Edge detection and linear feature extraction using 2-D random field model", *IEEE Trans. Patt. and Mach. Intell.*, PAMI-11, 1989, pp. 84-95.

[3] Venkateswar V. and Chellappa R., "Extraction of straight lines in aerial images", *SIGNAL PROCESSING V : Theories and Applications*, Elsevier Sc. Pub. B. V., 1990, pp. 1671-1674.

[4] Giraudon G., *Chaînage Efficace de Contour*, Technical Report 605, INRIA, fév. 1987.

[5] Canny J. F., "A computational approach to edge detection", *IEEE Trans. Patt. and Mach. Intell.*, PAMI-8, 1986, pp. 679-698.

[6] Rungsunseri Y. and Chehikian A., "A real time system for extracting edges and lines in images", *The 7th Scandinavian Conference on Image Analysis*, Aalborg, 13-16 Aug. 1991.

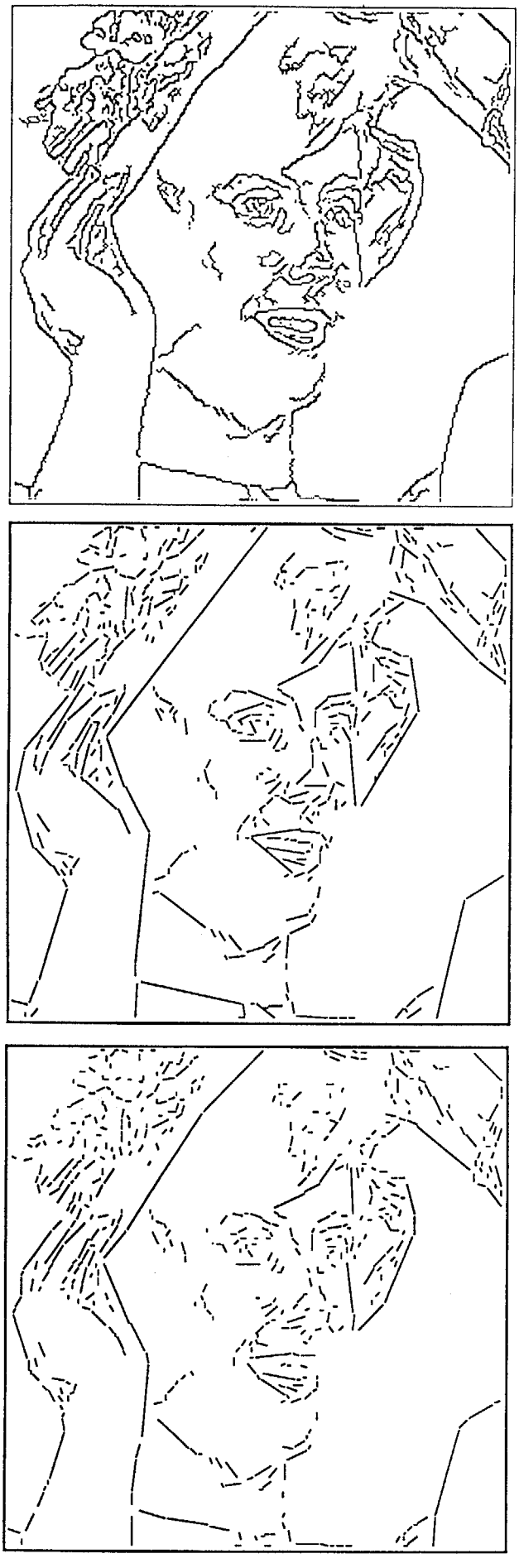

Fig.4 *Image de femme* ***a)*** *image de contour* ***b)*** *résultat avec 8 codes* ***c)*** *résultat avec 16 codes.*

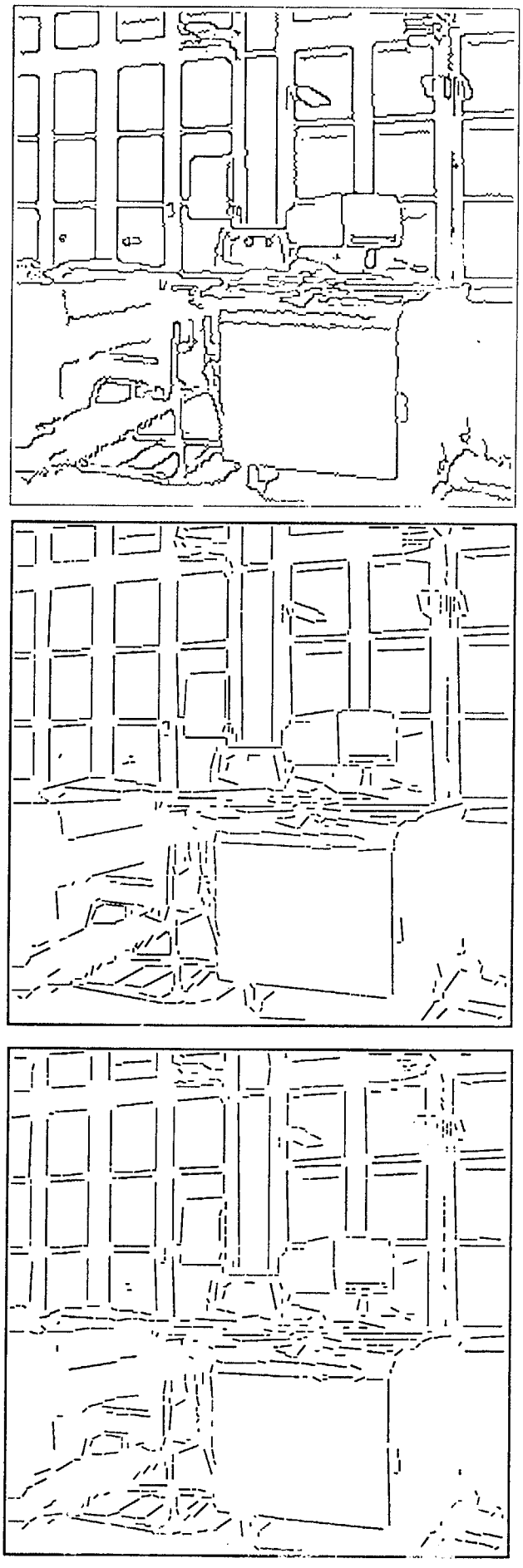

Fig.5 *Image de bureau* ***a)*** *image de contour* ***b)*** *résultat avec 8 codes* ***c)*** *résultat avec 16 codes.*